Bootstrap, T-SQL, XAML Anti-Patterns, Xamarin, Legal: Who Owns the Code?

CODE

SEP
OCT
2015

codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 5.95 Can $ 8.95

wtf.js?!?*

*well that's fabolous.js

Hadoop on Windows

Create Virtual Machines in the Cloud with Azure Skyline

Maximize Mobile Engagement with Interactive Onboarding

Features

Columns

Departments

US subscriptions are US $29.99 for one year. Subscriptions outside the US pay US $44.99. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards are accepted. Bill me option is available only for US subscriptions. Back issues are available. For subscription information, send e-mail to subscriptions@codemag.com or contact customer service at 832-717-4445 ext 028.

Subscribe online at codemag.com

CODE Component Developer Magazine (ISSN # 1547-5166) is published bimonthly by EPS Software Corporation, 6605 Cypresswood Drive, Suite 300, Spring, TX 77379 U.S.A. POSTMASTER: Send address changes to CODE Component Developer Magazine, 6605 Cypresswood Drive, Suite 300, Spring, TX 77379 U.S.A.

Canadian Subscriptions: Canada Post Agreement Number 7178957. Send change address information and blocks of undeliverable copies to IBC, 7485 Bath Road, Mississauga, ON L4T 4C1, Canada.

Hands-On for Great Code

The book you see in Figure 1 is called "Running MS-DOS" by Van Wolverton and it nearly ended my software development career before it even began. In 1988, I moved to Bend, OR and decided that it was time for this 19-year-old prodigy to enter college and prepare to achieve his lifetime goal of becoming a writer. In the first term, I took four classes: English Composition 101, Desktop Publishing, Introduction to Business Applications, and Editing. It was during this first term that I came to the realization that I liked the computer stuff more than the writing stuff (I bet Melanie will love that statement [Ed: <throat clearing>]), so I decided to sign up for more computer classes. The two I recall registering for were: Advanced Business Applications and Intro to MS-DOS. The Advanced Business Applications taught us how to write macros using Lotus-1-2-3 and WordPerfect. Intro to MS-DOS taught us...well...how to use MS-DOS. It's this second class that nearly ended my love for the computer classes. This is why...

Every week, we went through a number of exercises from the book. For instance, we learned the **dir** command. We used the **dir** command to list the contents of a directory and to show the date and time on files in a directory. We learned how to use wildcards to search for specific files. My strategy was to read, re-read, and re-read the chapter again in an attempt to remember the confusing syntaxes of slashes, back-slashes, wild cards, switches, etc. Then I'd take the test, which typically resulted in a CRASH and BURN operation.

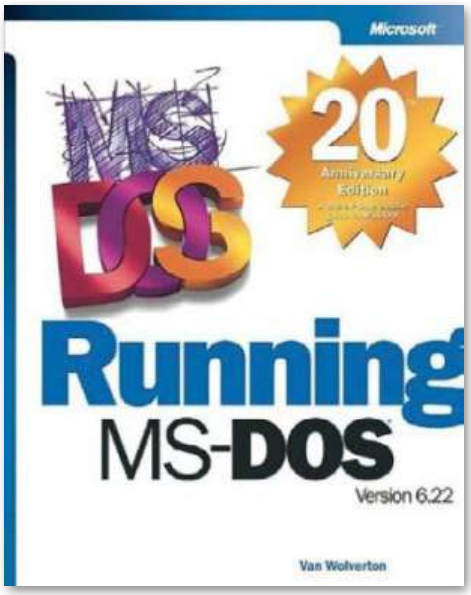


Figure 1: Van Wolverton's "Running MS-DOS" was life changing, but not how you think.

I was barely eking out 60s and 70s on the tests. What the heck? I read this stuff up, down, and sideways. How could I be failing? I went to my instructor with my frustrations and he asked, "Are you doing all the labs?" I told him that I was doing some of them but that I was reading the heck out of the stuff in the text. He said, "You need to focus more on the labs and less on the book." I promised to give it a try.

From that point on, I spent a lot more time in the lab and a lot less in the book. You know what happened? All of this MS-DOS stuff stuck. Finally, I was on track and began acing all my exams. I was off to the races. Next term, it was Advanced MS-DOS (batch files, baby) and Intro to Database Development (dBase III for the win). Sometimes a single discussion can be a life changer and the chat with my advisor was definitely that.

The point of this story is simple: Software development is an applied science. You can't become a software developer by reading a book (or Stackoverflow); you need to apply this stuff. It's applying the science that makes it stick. In the last issue's editorial, I focused on small life lessons that have helped me through my career. One I overlooked was the "applied" part of the science. This last month, I reached a breakthrough with my new developer. I got annoyed that some stuff we'd been working on hadn't been sticking. So I asked him the same question my advisor asked me: "Have you been writing code to practice the stuff?" His answer was just like mine: "I've been reading the heck out of it, so I don't understand what's going wrong." I told him to stop reading and start writing. You know what happened? He started doing that and is getting better steadily.

This is a lesson for you as well. You can read books, blog posts, magazine articles (keep reading CODE Magazine, please), and watch training videos, and yet still not understand the impact of a technology in your development life. It's not until you build an application (large or small) using a particular tool, technique or technology that you will really get it.

Over the last 20 years, I've written no fewer than four versions of my SQL-Server Stored Procedure Middleware. First it was in Visual FoxPro (two versions), then Visual Basic 6, then Visual Basic .NET and finally, in 2015, using C# with POCOs (Plain Old CLR Objects). Like all good developers, I spent time experimenting with different techniques, prototypes, and concepts. The first cut was very simple and handled mapping stored procedures to lists of objects. I immediately went to work implementing this first cut in production code. You know what happened? Some of it worked and some of it didn't. I improved the failing portion of the framework before moving on to doing CRUD operations on single tables. Then I went to work putting that into production as soon as possible. Rinse, lather, repeat. By putting this framework to real use, it got better and better. Without that technique, it would have worked only in theory and not necessarily in real life.

There's an old military saying that goes, "No battle plan survives contact with the enemy." This is a perfect statement when it comes to the applied aspects of software development. Remember: "Your designs aren't valid until they've been implemented and faced contact with the enemy (user)"!

Rod Paddock
CODE

RAVENDB

Safe by Default, Optimized for Efficiency

Data Made Simple

Following this simple rule, over the past few years, we have developed an optimized, high-performance Document Database with ACID transaction support.

To address large systems' needs, with availability and high-performance in mind, we have created a robust replication mechanism that allows you to quickly and more efficiently process requests and ensures that your system is always up and running.

The replication cluster can be composed of multiple nodes that are working together to create a single database that spans across multiple servers. Those nodes can have various relationships, from master-slave to geo-distributed, multi-master configuration, depending on the individual needs.

Our client API automatically detects cluster topology and switches to secondary nodes if server failure is detected or can take advantage of extra nodes during normal operation and use them to load-balance data processing.

www.ravendb.net

Proudly Developed by

Hibernating Rhinos

Manager's Corner: Recharge!

"It's just paint," I said by way of explanation. That seemed to make a lot of sense to him and my longtime friend eventually relaxed in the passenger seat. The sun was shining, the road was perfect, and I'd driven this winding stretch of road hundreds of times, so I'd been taking the straight path through the turns and it hadn't sat well with him until just that moment. I told him that I understood why the lines were painted on the road and had there been a car in either direction I'd have been in my lane, but there wasn't. So I'm a bit of a rule breaker. Not the kind who thinks rules are meant to be broken, but the kind who follows the *intent* of the law. I see things a bit differently than most. With the comment about paint, my friend understood that I wasn't violating any of the reasons that the paint had been put on the road, just ignoring, well, paint. Had I been pulled over, I would have taken the ticket without complaint. I couldn't argue that driving this way was legal, only that it was safe and enjoyable. But I usually do follow the rules, even to the letter of the law and at work, one of the most important rules is that when someone is on vacation, they're ON VACATION. I believe that a "working vacation" is not a vacation, it's just a change of scenery.

As a manager, I know that vacations are incredibly important, and not just for the person on vacation. Team dynamics change when a team member is out. Teammates pitch in. They step up their game a bit, knowing that these are the people who will cover for them when their own vacation comes around. Sometimes they need a little prodding, but nobody wants to spoil a good thing for everyone. People who don't often work together need to work together more. Some learn new skills to help pick up the slack. There is team building going on. And let's not forget that when teammates do come back from vacations, they're refreshed and recharged. Teams need to recharge periodically and having someone fresh off of vacation recharges the whole team by bringing a little more energy and a little more to talk about at the water cooler.

The funny thing about having real vacations, if you're lucky enough to work with a team who will really give you one, is that you never realize how much you need one until after you've taken it. We're all very good at plowing ahead and carrying on and getting the job done, and while we all like to have some time off, we tend to lose sight of the very reason we all need vacations in the first place: Perspective. The Oxford English Dictionary says perspective is: "true understanding of the relative importance of things; a sense of proportion." Our perspective, indeed our sense of reality, is shaped by where we're located (mentally at least) and how things look from there. Real perspective comes from looking at things from somewhere else, preferably some "higher ground," away from our everyday lives and all of the places we can see from there. A change in geography helps, but most importantly we need to climb up out of the well-worn grooves of our everyday lives. Out of what we do, what we think about, and what we see around us. For me, it takes about two full weeks before I even start to see over the ruts of my everyday life. Anything less than two weeks off is just a rest for me and not a vacation.

If you aren't on a team that values vacations this way, you should start building this culture. What good is being a manager if you can't influence and improve your team? Here are two simple rules for your team to get you started:

Vacation rule #1: When sending a teammate off to vacation, tell them in no uncertain terms, "Don't call and don't check your email. We've got you covered. We'll keep you off of any emails you don't really need to know about and we'll CC you on anything you need to know WHEN YOU GET BACK. We have your cell phone number and if you *did* forget to give someone the password that the success of the entire project hinges on before you left, we'll call you, ask for the password, and then say goodbye. Otherwise, don't expect to hear from us. Enjoy, have fun, sleep late, do cool things, take pictures. Go on vacation!"

Vacation rule #2: Most people on vacation will try to check in. Any response more detailed than, "Everything is fine. Can't wait to hear about your vacation when you get back," is too much. It's great to feel an obligation to the team, but it's also great that you can trust your team enough to let them handle work for a few weeks. If the vacationer doesn't get a proper vacation, then neither they nor the team will get the benefits. The world is (probably) not going to fall apart in a few short weeks no matter who is on vacation. If it does, then you haven't got a team in the first place.

As I write this, I'm sitting on the lanai, coffee cup beside me, staring out at the vast, blue Pacific Ocean. I'm just over two weeks into my vacation and really starting to fall into it. I wrote an email last week to a co-worker asking how things were going and suggesting we have a Skype call. Like I said, I am a rule breaker. When I get back, I have to remember to thank him for ignoring that email and handling everything for me, but right now I have to get down to the beach. Surf's up and when the waves are good, you have to get in the water. Those are the rules of surfing after all and, well, rules are rules.

Mike Yeager
CODE

Get back to writing code, not reports

Seamlessly and rapidly embed self-service reports, dashboards and visualizations

Izenda BI & Analytics

- Available in C# and Visual Basic for MVC and Web Forms
- Resides securely behind your firewalls and protocols
- Inherits your existing security model
- Rapid integration in cloud, hybrid and on-premise applications
- Rich, fully responsive and customizable front end

Get a free trial

See how Izenda can work inside your application so you can get back to working on yours.

IZENDA
Integrate • Accelerate • Innovate
BUSINESS INTELLIGENCE

Visit us at www.izenda.com to learn more and download your free trial.

A Data-Driven Menu System for Bootstrap

Bootstrap makes it easy to create a nice looking menu system (see **Figure 1** and **Figure 2**). Instead of coding all your menus within your shared layout file, make your menus dynamic by storing them in an XML file or a database table. In this article, you create a simple one-line menu system using two C# classes and a little bit of Razor code in an MVC page. The sample in this article shows you how to build a hierarchical menu structure to be used with drop-down menus. You'll also see how to store and retrieve the menus from an XML file using LINQ to XML and a little bit of recursion.

Paul D. Sheriff
PSheriff@pdsa.com

Paul D. Sheriff is the President of PDSA, Inc. (http://www.PDSAServices.com). PDSA develops custom business applications specializing in Web and mobile technologies. PDSA, founded in 1991, has successfully delivered advanced custom application software to a wide range of customers and diverse industries. With a team of dedicated experts, PDSA delivers cost-effective solutions, on-time and on-budget, using innovative tools and processes to better manage today's complex and competitive environment. Paul is also a Pluralsight author. Check out his videos at http://www.pluralsight.com/author/paul-sheriff.

Menu Classes

For any menu system, you need a minimum of two properties: **Title** and **Action**. **Title** is the text to display to the user and **Action** is the anchor tag to take the user to the page represented by the menu. As with most everything in .NET, you use a class to represent a menu. In the code snippet below, you'll find a class called **BSMenuItem** that has these two properties.

```
public class BSMenuItem
{
  public string Title { get; set; }
  public string Action { get; set; }
}
```

Create a single instance of the BSMenuItem class for each menu you wish to display on the screen. As shown in **Figure 1,** you have four menus to display, so you create four instances of this class with the appropriate titles filled in Home, Maintenance, Reports, and Lookup.

A Menu Manager Class

To build this collection of BSMenuItem objects, create a class called **BSMenuItemManager** (as shown in **Listing 1**). This class contains a single property called Menus, which is a generic list of BSMenuItem objects. A **Load** method creates all four of the menus shown in Figure **1**. In the constructor, create a new instance of the generic list of BSMenuItem objects. In the Load method, create a new object, fill in the **Title** and **Action** properties, and then add that new object to the Menus collection.

The Menu Controller

Create a new MVC page called **DataDrivenMenu**. In the controller for your new page, create a new instance of the BSMenuItemManager class. Call the **Load** method to build the collection of BSMenuItem objects. Pass the instance of the manager object to the View so you can use the Menus property from the .cshtml page.

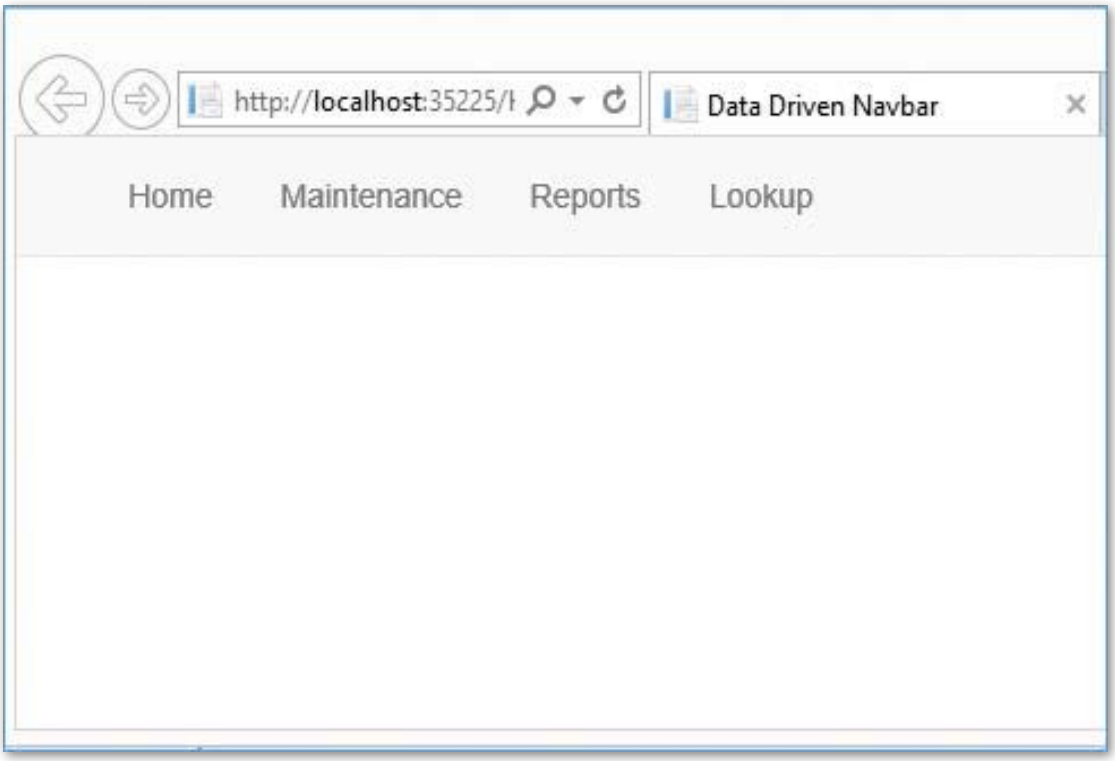


Figure 1: A simple data-driven menu system

DOMAINS | MAIL | HOSTING | eCOMMERCE | SERVERS

NEXT GENERATION 1&1 CLOUD SERVER

TRY IT FOR 1 MONTH - FREE!
Then starting at $9.99 per month*

Powered by intel Cloud Technology

Top performer CLOUD SPECTATOR

EASY TO USE – READY TO GO

The new 1&1 Cloud Server offers all the advantages of dedicated hardware performance combined with the flexibility of the cloud!

FLEXIBLE & AFFORDABLE

Customized configuration

- SSD, RAM and CPU can be adjusted independently, flexibly and precisely
- **NEW:** Pre-configured packages available

Windows

Transparent costs

- Billing by the minute
- Clearly structured cost overview enables efficient planning and management
- No minimum contract term

EASY & SECURE

1&1 Cloud Panel

- Innovative, user-friendly interface with smart administration

Security

- Built-in firewall to protect your server from online threats
- Backups and snapshots to prevent accidental data loss
- High-performance 1&1 Data Centers are among the safest in the US

ALL-INCLUSIVE

Best performance

- Unlimited traffic
- Premium SSD with the highest performance
- Private networks, professional API, load balancers, firewalls and more- all easy to configure
- Ready-to-use applications including WordPress, Drupal™ and Magento®
- Powered by Intel® Xeon® Processor E5-2683 V3 (35M Cache, 2.00 Ghz)

1 TRIAL TRY FOR 30 DAYS ★★★★★

1 MONTH FLEXIBLE PAYMENT OPTIONS ★★★★★

1 CALL SPEAK WITH AN EXPERT 24/7 ★★★★★

1 (877) 461-2631

1&1®

1and1.com

*1&1 Cloud Server is available free for one month, after which regular price of $9.99/month applies. No setup fee is required. Visit 1and1.com for full offer details, terms and conditions. Intel, the Intel Logo, Intel Inside, the Intel Inside logo, Intel. Experience What's Inside are trademarks of Intel Corporation in the U.S. and/or other countries. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are property of their respective owners.

```
public ActionResult DataDrivenMenu()
{
  BSMenuItemManager mgr =
     new BSMenuItemManager();

  mgr.Load();

  return View(mgr);
}
```

The .cshtml Page

The first thing you do in your .cshtml page (**Listing 2**) is set up the model for the page. The model is the instance of the BSMenuItemManager that you created in the controller. The model contains the Menus property, which contains the collection of BSMenuItem objects created in the **Load** method. Add the appropriate HTML and Bootstrap classes to create the container around the menu items that you need to build. Use a **foreach** loop to iterate over the collection of menu items. Within the **foreach** loop, build a **<li>** tag with an anchor tag inside each. The **href** attribute is set to the **Action** property and the text within the anchor is set to the **Title** property. This is all it takes to build a simple one-line menu system.

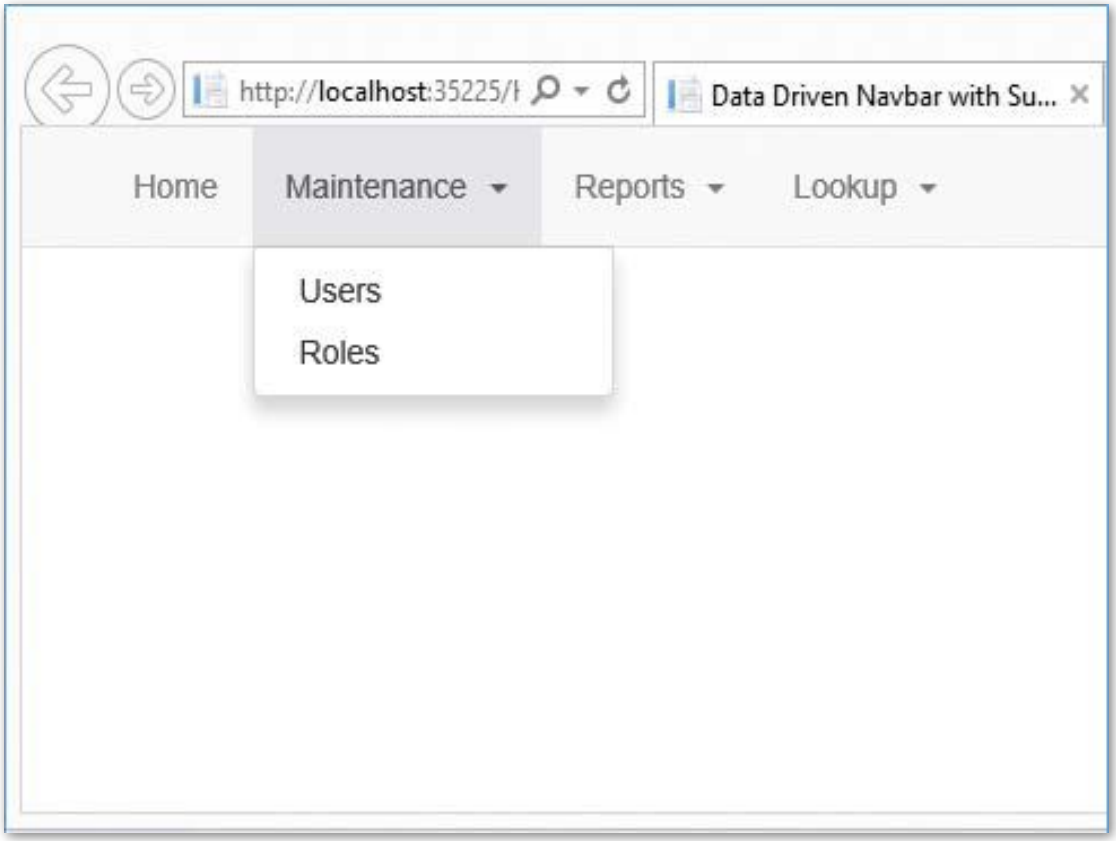


Figure 2: Drop-down menus are easy to do with Bootstrap.

Sample Code

You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select PDSA Articles, then select "Code Magazine—A Data-Driven Menu System for Bootstrap" from the drop-down list.

Listing 1: A hard-coded approach to a simple menu system.

```
public class BSMenuItemManager
{
  public BSMenuItemManager()
  {
    Menus = new List<BSMenuItem>();
  }

  public List<BSMenuItem> Menus { get; set; }

  public void Load()
  {
    BSMenuItem entity = new BSMenuItem();

    entity.Title = "Home";
    entity.Action = "/Home/Home";
    Menus.Add(entity);

    entity = new BSMenuItem();
    entity.Title = "Maintenance";
    entity.Action = "/Maintenance/Maintenance";
    Menus.Add(entity);

    entity = new BSMenuItem();
    entity.Title = "Reports";
    entity.Action = "/Reports/Reports";
    Menus.Add(entity);

    entity = new BSMenuItem();
    entity.Title = "Lookup";
    entity.Action = "/Lookup/Lookup";
    Menus.Add(entity);
  }
}
```

Listing 2: A foreach loop creates a bootstrap menu.

```
@model BSMenuItemManager

@{
  ViewBag.Title = "Data Driven Navbar";
}

<nav class="navbar navbar-default navbar-static-top"
     role="navigation">
  <div class="container">
    <ul class="nav navbar-nav">
      @foreach (BSMenuItem item in Model.Menus)
      {
        <li>
          <a href="@item.Action">
            @item.Title
          </a>
        </li>
      }
    </ul>
  </div>
</nav>
```

Listing 3: A self-referencing object is ideal for drop-down menus.

```
public class BSMenuItem
{
  public BSMenuItem()
  {
    ParentId = 0;
    Menus = new List<BSMenuItem>();
  }

  public int MenuId { get; set; }
  public int ParentId { get; set; }
  public string Title { get; set; }
  public int DisplayOrder { get; set; }
  public string Action { get; set; }

  public List<BSMenuItem> Menus { get; set; }
}
```

Hierarchical Menus

In many websites, you need drop-down menus (**Figure 2**) instead of single-line menus. Using Bootstrap, you can accomplish this quite easily. In order to use a data-driven approach, you need to modify the BSMenuItem class (**Listing 3**) so it holds a collection of other BSMenuItem objects. A few other properties need to be added as well. You need a **MenuId** to be used as a primary key. A **ParentId** property is set to the MenuId value of another menu to indicate that this menu is a sub-menu. Set the ParentId property to zero to signify that this menu is a top-level menu. The last property is **DisplayOrder** so you can sort the menus in a specific order before you display them.

Store Menus in an XML File

Instead of hard-coding the menus, let's put the menus into an XML file. The XML reflects the new structure defined in the BSMenuItem class. The next code snippet shows a single node of the XML file. The **ParentId** is set to zero to represent that this menu is one of the top-level menus.

```
<Menus>
  <Menu>
    <MenuId>1</MenuId>
    <ParentId>0</ParentId>
    <Title>Home</Title>
    <DisplayOrder>10</DisplayOrder>
    <Action>/Home/Index</Action>
  </Menu>
</Menus>
```

In **Figure 3**, you see the relationship between a **ParentId** and a **MenuId** within the XML file. If the **ParentId** is not zero and is equal to the same value as a **MenuId**, then that menu is one that is a drop-down under the menu with that **MenuId** value.

Figure 3: Self-referencing helps avoid multiple XML files.

Load the Self-Referencing Menus

Modify the Load method of the BSMenuItemManager class to use an **XElement** class to load the XML file (**Listing 4**). Pass the collection of all menus to a new method called **LoadMenus**. This recursive method is passed an XElement object and a value of the menus that have a **ParentId** that match that value. The Load method passes the value zero to only select those nodes from the XElement object passed in where the **ParentId** is equal to zero. Zero represents all of the top-level menus only.

XML Extension Methods

To make XML processing easier, create an extension method for use with XElement objects (**Listing 7**). This extension method is a generic method in that you pass it the data type you wish to return from this method. This method checks to see if the element passed in is null or has an empty value. If there's a value, that value is converted into the generic type passed in using the Convert class' ChangeType method. Be careful in using the ChangeType method as it won't work with nullable types.

There's an optional parameter you can pass to this method, and that's the default value to return if the value in the XElement object is null or empty. For example, you could pass in a minus one (-1) for any integer value. If the element is blank then a -1 is returned. If you don't pass anything into this parameter, then the C# keyword default calculates the default value for the data type passed in.

This class only handles XElement objects, which means your XML must be all element-based. If you have attributes that you are reading from your XML, you'd add another method to handle XAttribute objects. The code is identical except that you use an XAttribute class instead of an XElement class.

In the LoadMenus method, LINQ to XML is used to select those nodes from the XElement object that match the **ParentId** passed in. Apply a **where** clause to perform the filtering and use an **orderby** to sort by the **DisplayOrder** element. A **select new BSMenuItem** statement creates a new object and maps the XML elements to each property of the new object. Notice the use of an extension method **GetAs**. This method is explained in the sidebar **XML Extension Methods**.

Notice the line of code within the **select** that sets the **Menus** property of the BSMenuItem class. A ternary operator is used to determine whether or not to recursively call the LoadMenus method again or to just create an empty list of BSMenuItem objects. If the **ParentId** is not equal to the **MenuId**, you need to check to see if there are any sub-menus that match the current **MenuId** in the XML node that is being processed. Pass that **MenuId** to the LoadMenus method and any sub-menus are loaded and returned into the **Menus** property.

```
Menus =
 (ParentId !=
    node.Element("MenuId").GetAs<int>() ?
  LoadMenus(menus,
    node.Element("MenuId").GetAs<int>()) :
  new List<BSMenuItem>())
```

Modify the Controller

The controller to call this new Load method in the BSMenuItemManager class must now change. You need to pass in the name of the XML file that contains the menus to the Load method. Use **Server.MapPath** to calculate the complete file path and name within your website. In the next code snippet, I created a file called **Menus.xml** that was under the folder \Xml.

```
public ActionResult DataDrivenMenu()
{
  BSMenuItemManager mgr =
    new BSMenuItemManager();

  mgr.Load(Server.MapPath("/Xml/Menus.xml"));

  return View(mgr);
}
```

Modify the .cshtml Page

To display a drop-down menu in Bootstrap, add additional attributes to your anchor tag and create a new **<ul>** tag with the collection of sub-menus. Add an **if** statement around the code you had before in the .cshtml page. Check the current menu item and see if there are any sub-menus in its **Menus** property. If not, use the same code to build the anchor tag that you used previously.

Listing 4: Using LINQ to XML and recursion to load hierarchical menus.

```
public class BSMenuItemManager
{
  public List<BSMenuItem> Menus { get; set; }

  public List<BSMenuItem> Load(string location)
  {
    XElement menus = XElement.Load(location);

    Menus = LoadMenus(menus, 0);

    return Menus;
  }

  private List<BSMenuItem> LoadMenus(XElement menus,
                                     int ParentId)
  {
    List<BSMenuItem> nodes = new List<BSMenuItem>();

    nodes =
      (from node in menus.Elements("Menu")
       where node.Element("ParentId").GetAs<int>()
             == ParentId
       orderby node.Element("DisplayOrder").GetAs<int>()
       select new BSMenuItem
       {
         MenuId = node.Element("MenuId").GetAs<int>(),
         ParentId = node.Element("ParentId").GetAs<int>(),
         Title = node.Element("Title").Value,
         DisplayOrder =
           node.Element("DisplayOrder").GetAs<int>(),
         Action = node.Element("Action").Value,
         Menus =
          (ParentId != node.Element("MenuId").GetAs<int>() ?
           LoadMenus(menus,
              node.Element("MenuId").GetAs<int>()) :
           new List<BSMenuItem>())
       }).ToList();

    return nodes;
  }
}
```

Listing 5: A nested foreach loop builds the sub-menus

```
else
{
  <a href="@item.Action"
     class="dropdown-toggle"
     data-toggle="dropdown">
     @item.Title 
     <span class="caret"></span>
  </a>
  <ul class="dropdown-menu">
    @foreach (BSMenuItem subitem in item.Menus)
    {
      <li>
        <a href="@subitem.Action">
          @subitem.Title
        </a>
      </li>
    }
  </ul>
}
```

Listing 6: This function strips any non-numeric characters from a string.

```
@model BSMenuItemManager

@{
  ViewBag.Title = "Data Driven Navbar with Submenus";
}

<nav class="navbar navbar-default navbar-static-top"
     role="navigation">
  <div class="container">
    <ul class="nav navbar-nav">
      @foreach (BSMenuItem item in Model.Menus)
      {
        <li>
          @if (item.Menus.Count == 0)
          {
            <a href="@item.Action">
               @item.Title
            </a>
          }
          else
          {
            <a href="@item.Action"
               class="dropdown-toggle"
               data-toggle="dropdown">
              @item.Title 
              <span class="caret"></span>
            </a>
            <ul class="dropdown-menu">
              @foreach (BSMenuItem subitem in item.Menus)
              {
                <li>
                  <a href="@subitem.Action">
                     @subitem.Title
                  </a>
                </li>
              }
            </ul>
          }
        </li>
      }
    </ul>
  </div>
</nav>
```

Listing 7: Use extension methods to simplify code.

```
public static class XmlExtensionMethods
{
  public static T GetAs<T>(this XElement elem,
                           T defaultValue = default(T))
  {
    T ret = defaultValue;

    if (elem != null && !string.IsNullOrEmpty(elem.Value))
    {
      ret = (T)Convert.ChangeType(elem.Value, typeof(T));
    }

    return ret;
  }
}
```

```
@if (item.Menus.Count == 0)
{
  <a href="@item.Action">
    @item.Title
  </a>
}
```

Next, add an **else** statement (**Listing 5**) to build submenus if there are submenus to display. Submenus in Bootstrap are built by adding the class of dropdown-toggle to the anchor tag of the top-level menu. Add **data-toggle="dropdown"** attribute to the anchor tag as well. Display the **Title** within the anchor tag, and next to the title, add a blank space and a span tag with the class of **caret** to display an arrow next to the menu.

Immediately below the anchor tag add a new **<ul>** tag and iterate through all submenus building the **<li>** tags for each menu item just like you did for your normal menus. You can see the complete .cshtml page in **Listing 6**.

Summary

In this article, you learned to build a Bootstrap menu system using C# classes and **foreach** loops. You also saw how to build a hierarchical menu system using LINQ to XML and a recursive method. Keeping menus in an XML file is convenient if you aren't using a database in your application. Using a self-referencing system of a MenuId and corresponding ParentId avoids the need to have multiple XML files.

Paul D. Sheriff

Legal Notes: Who Owns the Code?

Although the question is simple, the answer can be quite complex. If you're a client who's engaged the services of a contractor, you're inclined to think that since you're paying for services that lead to code, you own the copyright to the code. You may, in fact, own the copyright, but not because you paid for the services. On the other hand, you may be a consultant and heard something like that in the absence of a contract, you own the copyright for the code you provided your client. You may, in fact, own the copyright, but not because there isn't a contract.

John V. Petersen, Esq.
johnvpetersen@gmail.com
codebetter.com/johnpetersen
@johnvpetersen

John is a graduate of the Rutgers University School of Law in Camden, NJ and has been admitted to the courts of the Commonwealth of Pennsylvania and the state of New Jersey. For over 20 years, John has developed, architected, and designed software for a variety of companies and industries. John is also a video course author for WintellectNOW. John's latest video series focuses on legal topics for the technologist. You can learn more about this series here: http://bit.ly/WintellectNOW-Petersen. If you are new to WintellectNOW, use promo-code PETERSEN-14 to get two weeks of free access.

So...who owns the code? Like most things legal, the simplest answer is "It depends." This article will provide you with the information to be able to answer this question.

DISCLAIMER: This and future columns should not be construed as specific legal advice. Although I'm a lawyer, I'm not *your* lawyer. The column presented here is for informational purposes only. Whenever you're seeking legal advice, your best course of action is to **always** seek advice from an experienced attorney licensed in your jurisdiction.

Avoid Confusion: Have a Contract that Spells Out Who Owns the Code

In my last article, I discussed employment agreements. One of the big items addressed in most employment and independent contractor agreements is intellectual property (IP) ownership. Often, the code and application delivered is regarded as *Work for Hire*. If that's the case, that settles the matter: The client who hired the contractor owns the copyright. Often, such agreements can take an overly broad view of what *Work for Hire* is. Often, such a definition can also include things that are not directly related to the project. This is why it's important for contractors to get as narrow a definition of *Work for Hire* as possible.

If you are a W-2 full-time employee, then by default and at the very least, regardless of whether there is any form of agreement, your employer owns what you do while on the clock, on the company premises, or using company knowledge and assets. Sometimes, employees sign agreements that broaden what the employer can claim ownership of. Like consultants, employees should strive to narrow that definition.

When you're not an employee, what happens when there's no contract? Or, what if there is a contract but there's no provision for intellectual property ownership As it turns out, in the absence of a contract, the question of IP ownership depends on whether or not you are, for legal purposes, considered an employee. Even if you're a 1099 consultant, you may be a *Statutory Employee*. If you're deemed to be a Statutory Employee, then just like a W-2, your *Employer* owns your work.

Are You a Statutory Employee?

I've already addressed cases where if someone is an employee, the employer owns the employee's work. An employee for these purposes is someone who's hired by the employer and receives W-2 wages. You may be wondering how in the world can someone be a consultant on one hand but be regarded as an employee on the other hand? For that answer, you need not look any further than our friends at the Internal Revenue Service (IRS). All 1099 wages are taxed differently than W-2 wages. For one thing, there's no employer match on the federal withholding taxes. And that's where the IRS's original 20-question test applies. Eventually, this test was reduced to an 11 factor test. Details about the test can be found here: http://www.irs.gov/Businesses/Small-Businesses-&-Self-Employed/Independent-Contractor-Self-Employed-or-Employee.

In a nutshell, the factors fall into these categories:

- **Behavioral:** Does the company control or have the right to control what the worker does and how the worker does his or her job?
- **Financial:** Are the business aspects of the worker's job controlled by the payer? (These include things like how the worker is paid, whether expenses are reimbursed, who provides tools/supplies, etc.)
- **Type of Relationship:** Are there written contracts or employee-type benefits (i.e., pension plan, insurance, vacation pay, etc.)? Will the relationship continue and is the work performed a key aspect of the business?

If it turns out that the client exerts a great deal of control over what you do, how you do it, where you do it, etc., then more likely than not, you're a statutory employee. Those factors vest ownership with the client/employer. Such a determination has consequences for the client/employer given that more likely than not, they weren't withholding and paying their share of federal taxes as they would be required to do for W-2 employees.

Watch Those Emails and Any Other Writing

Courts will do whatever they can to find a contract that embodies the intent of the parties. Sometimes, courts infer a contract from the conduct and communications between the parties. Even if you have a contract, emails and other writings, under certain circumstances, can alter the landscape. As the famous former mayor of Philadelphia once quipped: "Don't ever write a letter. If you get one, don't ever throw it away."

What's the Bottom Line?

The issue breaks down as follows:

- **If you are an employee:** At the very least, your employer owns the code.

- Depending on your employment arrangement, what your employer owns may be broader than what you think.
- **If you are a consultant (1099) and you have a contract that specifies that what you do is work for hire:** The client owns your work.
- Like an employment situation, depending on the arrangement, what the client owns may be broader than what you think. The terms of the contract control who owns the code.
- If neither of the above apply, the question of ownership turns on whether you are an employee for statutory purposes. That question is answered by applying the IRS test.

> In the absence of actual employment or a contract or anything else that explicitly addresses ownership, the issue of ownership depends on whether the person writing the code is considered a statutory employee.

The best advice I can give here is that if you are an independent contractor, have a contract that specifically outlines who owns what! If you have anything less, you're throwing caution to the wind. Understand what the rules are and to have the ability to cry foul when somebody attempts to claim ownership when no such standing exists.

John V. Petersen

Getting Up to Speed

If you haven't read my previous article on employment agreements, please do so, as it will give you more context for this article.

The one question you may still have is how in the world does the federal government's taxing authority have any standing on intellectual property issues? Often, courts, through the briefs presented by the litigants, will use other areas of law as guidance. For more background on this issue, please refer to this whitepaper drafted by the Joint Committee on Taxation: http://www.irs.gov/pub/irs-utl/x-26-07.pdf.

A more current example of where the issue on employee vs. contractor is at issue is the recent Uber case in California: http://www.scribd.com/doc/268947596/Uber-Filing. Although this article is about intellectual property rights, it's important to note that many rights, obligations and liabilities can apply based on the determination of whether an individual is deemed to be an employee or an independent contractor. Cases like Uber in California can find their way to be precedent-setting in other jurisdictions.

More about Xamarin Pages

In the last issue, I discussed the fundamental blocks of the page model. For those who missed that article, here's a 10,000-foot view of the page framework. Every UI development system has the notion of screens: separate units of UI real estate that divide the application into work areas. They go by different names; you may know them as views, forms, dialogs, or pages.

Walt Ritscher
waltr@scandiasoft.com
xamlwonderland.com
@waltritscher

Walt Ritscher's enthusiasm for crafting software interfaces blossomed early, first surfacing while coding on a borrowed computer. Now he travels the world speaking at software conferences and teaching a diverse portfolio of programming topics. On the consulting side, he works with customers like Microsoft, HP, Intel, and ExxonMobil and enjoys being part of the Wintellect consultant group.

His *HLSL and Pixel Shaders for XAML Developers* book is available from O'Reilly Media. He has nearly fifty hours of video training courses available in the WintellectNow, Lynda.com, and Udemy catalogs. He specializes in live, on-site training and can tailor a custom class for your needs.

His current UI obsession revolves around the Windows 8 Metro, Mobile, ASP.NET MVC, and WPF APIs. This year, he's avidly learning as much as he can about Xamarin. Walt is also an MVP and the author of the free Shazzam Shader Editor at Shazzam-tool.com.

Whatever the name, they serve a useful purpose: keeping the UI bits separated into logical sub-areas of the application.

Pages are the top-level elements in the Xamarin UI. Think of pages as the screens in your app. Most of the time, a page takes up the entire screen of the device. However, pages can have sub-pages embedded within their content. In that case the sub-pages are more like a composite user control. If you're an experienced mobile developer, a Xamarin Page corresponds to iOS View Controller, Android Activity, or Windows Phone Page.

Navigation: Moving Between Pages

When faced with building an app containing multiple pages (see **Figure 1**), you have to determine your navigation strategy. There are plenty of questions to ask your stakeholders and team including:

- How do the users move between pages?
- Does the content of the next page depend on choices made on current page?
- How is state passed between pages?
- Do you need to track the navigation history?
- A healthy app should be forgiving; how will you handle undo and associated navigation?

Once you have the strategy in place, it's time to choose the navigation's UI. There are plenty of patterns available. Here's a partial list:

- Buttons on the page that navigate to another page
- Drop-down menus
- Pagination links
- Breadcrumb links
- Appbar
- Tabs
- Carousel

This article is not the place to discuss the UX principles for good navigation. But it is the place to show what navigation tools are available in Xamarin.Forms.

The Page Types

There are a number of page types available in the Xamarin library (see **Figure 2**). Let's look at the ones that are helpful for navigation purposes.

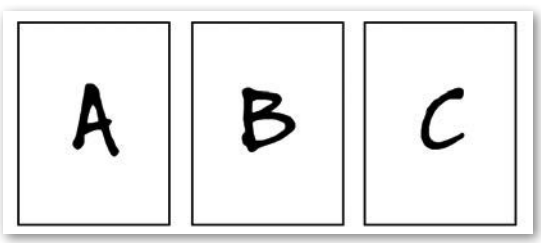


Figure 1: Three pages

In this article, I'll explore the **TabbedPage** and **CarouselPage**. The TabbedPage and CarouselPage are containers for child pages. They host the child content (usually contained in a ContentPage) and provide the UI affordances to enable the user to switch between pages. The main difference between the two page types is how they encourage users to move between pages. The TabbedPage uses the familiar tab metaphor; you switch between child pages by touching a tab. The CarouselPage depends heavily on the left-right swipe gesture to move between child pages.

That leaves the **NavigationPage**. Its main benefit is to provide a navigation stack, which holds the page history, and a navigation appbar for Android and iPhone devices. I'll explore the NavigationPage in another article.

To learn more about the **Page** base class, the **ContentPage** or **MasterDetailPage** check out the last article in this series: Xamarin Pages: The Screens of an App (July/August 2015 http://www.codemag.com/Article/1507101).

Tabbed Content

Using tabs for application navigation has a long history in the PC world. You've seen examples of it on your computer in dozens of programs. The basic concepts are shown in **Figure 3**.

> Luckily for us, the Xamarin TabbedPage takes care of providing the right look for each platform.

Tabbed UI usually has a flat section on one side of the UI (most commonly the top side) that contains a list of tab items. Each tab has text describing the page. Some implementations mimic the tabbed look of office tab folders; others choose a minimalist look showing a set of horizontal links. Good implementations indicate which tab is selected, so the user can tell at a glance where they are in the app.

Use a tabbed interface to provide a readable list of page choices. Because of the limited width of mobile devices, a tabbed interface is not a good choice if the tab names are long or there are more than three or four pages.

Each mobile platform has different UI metaphors for the tab appearance. Luckily for us, the Xamarin TabbedPage takes care of providing the right look for each platform.

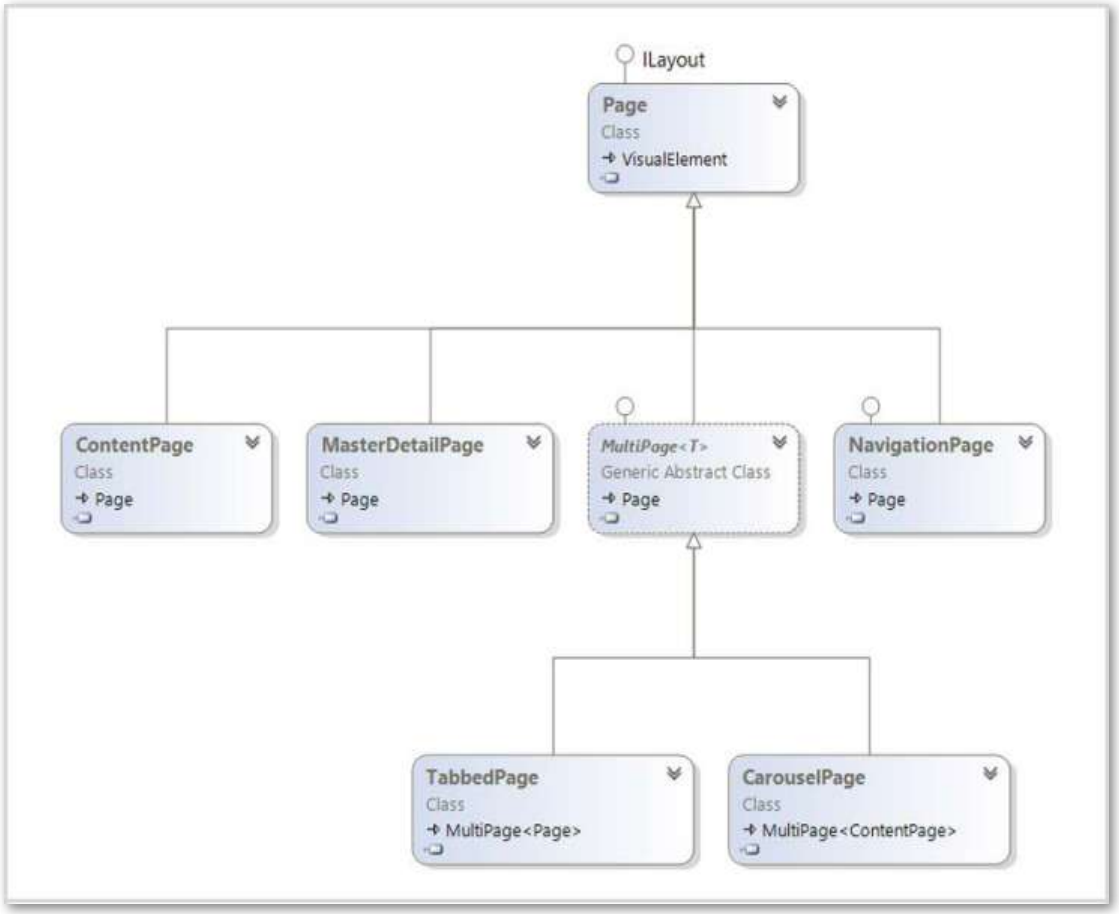


Figure 2: The Page object diagram

Assumptions

This article is a continuation of my Xamarin Pages article in the last issue of CODE Magazine (July/August 2015 http://www.codemag.com/Article/1507101. I recommend that you study that article to gain insights into Page basics. Also, if you're new to Xamarin, be sure to read the past issues of CODE Magazine to learn about the core parts of Xamarin apps, see how to install Xamarin, make a basic project, and review the Xamarin object structures. I favor the Xamarin integration in Visual Studio, so that's what I'm using for this article. I'm working in Visual Studio 2013 but there are Xamarin versions available for Visual Studio 2012/2015. There's also Xamarin Studio, should you prefer another IDE.

Making the TabbedPage

The TabbedPage is simple to use. In XAML, add each page as child elements, as shown in this code snippet.

```
<TabbedPage
  xmlns="…"
  xmlns:x="…"
  xmlns:local="clr-namespace:Nav;assembly=Nav"
  x:Class="Nav.WeatherTabbedEmpty">

  <ContentPage Title="Maps"
               BackgroundColor="#FF60337F"/>
  <ContentPage Title="Forecasts"
               BackgroundColor="Teal"/>
  <local:SingaporePage />
</TabbedPage>
```

The first two tabs are empty, created entirely in XAML. Here, I'm setting the BackgroundColor and Title properties. Setting the title property is important as it's shown on the tab UI.

The third tab page is more complex; I'm adding it via a custom XML namespace. The Singapore page is defined in a separate XAML file (see **Listing 1**).

Finally, I'm adding a fourth page dynamically in the code behind page, as shown in the following snippet.

```
namespace Nav {
  public partial class WeatherTabbed :
                                  TabbedPage {
    public WeatherTabbed() {
```

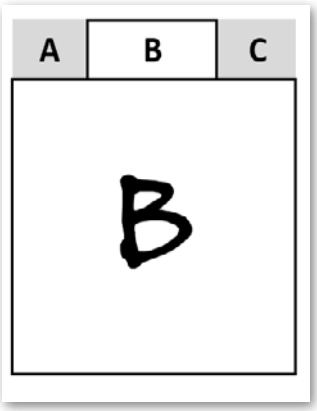


Figure 3: Tabbed Interface

Listing 1: SingaporePage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Nav.SingaporePage" Title="Singapore">
  <ContentPage.Resources>
    <ResourceDictionary >
      <Style  TargetType="Label">
        <Setter Property="VerticalOptions" Value="Center" />
        <Setter Property="HorizontalOptions" Value="End" />
        <Setter Property="FontSize" Value="20" />
        <Setter Property="TextColor" Value="Black" />
      </Style>
    </ResourceDictionary>
  </ContentPage.Resources>

  <StackLayout BackgroundColor="#FFd700" Padding="10">
    <Label Text="Singapore Weather"  FontAttributes="Bold"/>
    <Label Text="High - 25C" />
    <Label Text="Sunny" />
  </StackLayout>
</ContentPage>
```

Listing 2: BarcelonaPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Nav.BarcelonaPage" Title="Barcelona">
  <ContentPage.Resources>
    <ResourceDictionary >
      <Style  TargetType="Label">
        <Setter Property="VerticalOptions" Value="Center" />
        <Setter Property="HorizontalOptions" Value="End" />
        <Setter Property="FontSize" Value="20" />
        <Setter Property="TextColor" Value="Black" />
      </Style>
    </ResourceDictionary>
  </ContentPage.Resources>
  <StackLayout BackgroundColor="#00a86b" Padding="10">
    <Label Text="Barcelona Weather"  FontAttributes="Bold" />
    <Label Text="High - 18C"   />
    <Label Text="Partly Cloudy"  />
  </StackLayout>
</ContentPage>
```

Figure 4: TabbedPage in Android

Figure 5: TabbedPage in iPhone

```
        InitializeComponent();

        this.Children.Add(new BarcelonaPage());
      }
    }
}
```

Check out the full XAML for the Barcelona page in **Listing 2**.

That's it! When the **WeatherTabbed** page is opened, this is what you see (**Figure 4**) on an Android device (running KitKat v.4.4).

On my Moto G phone (720p, 4.5 inch display), the four tabs can't all fit on the screen at the same time. See how the Maps tab is scrolled off the left edge of the screen? The Android tab implementation supports swipe-left/swipe-right to move to hidden tabs.

The tab text defaults to all caps. On my device, there's a blue bar below the selected tab to indicate that it's the active tab.

Here's the same UI (**Figure 5**) running on an iPhone 5S (iOS 8.1).

On this device, the tabs are located on the bottom of the screen. As you can see, all of the tabs fit on the screen. The selected tab is indicated with a different color font, blue in this screenshot.

Finally, here is the same UI running on a Windows Phone (**Figure 6**).

> One drawback of carousels on touch devices is that the user may not know that a swipe gesture is available for navigation.

This phone uses a giant typeface for the tab text. As a result, only two tab items fit on this screen (1080p, 6- inch display). It does support swipe-left/swipe-right gestures for the tab bar. Windows phone also wraps the content, so I can see part of the Maps tab to the right of the Barcelona tab.

Let's have a round of applause for the Xamarin team; they make the tabbed UI work in the familiar ways on each device.

The CarouselPage

The side-swipe gesture is a fundamental part of modern mobile interfaces. This makes it easy to do away with the tabbed metaphor (where the user has to touch the tab portion of the screen in order to navigate) and provide a gesture-based way to move between child pages (see **Figure 7**).

One drawback of carousels on touch devices is that the user may not know that a swipe gesture is available for navigation. Some implementations solve this issue by showing a small slice of the side pages when looking at the current page (see **Figure 8**). Because most mobile developers prefer to maximize screen real estate, many carousel controls don't show the off-screen page edges.

Making a CarouselPage

Using a CarouselPage is a snap! The XAML/code is nearly identical to the TabbedPaged code; just change the page type to CarouselPage, as shown in the following XAML snippet.

```
<CarouselPage
  xmlns="…"
  xmlns:x="…"
  xmlns:local="clr-namespace:Nav;assembly=Nav"
  x:Class="Nav.WeatherCarousel">

  <ContentPage Title="Maps"
               BackgroundColor="#FF60337F"/>
  <ContentPage Title="Forecasts"
               BackgroundColor="Teal"/>
  <local:SingaporePage />
</CarouselPage>
```

Of course, the C# code needs a few minor tweaks.

```
namespace Nav {
  public partial class WeatherCarousel :
                                TabbedPage {
    public WeatherCarousel() {
      InitializeComponent();

      this.Children.Add(new BarcelonaPage());
    }
  }
}
```

Figure 6: TabbedPage on Windows Phone

Here's a screenshot of the CarouselPage on Android (**Figure 9**).

The iOS version is virtually identical so there's no need for a screenshot. The Windows phone implementation is interesting and slightly different (see **Figure 10**). Notice how it provides a UI clue that there are other pages by showing a slice of the next page on the right side of the current page.

Using a Data Source and Data Template

If you're a data binding fan, you'll be happy to learn that the TabbedPage supports binding of the child page collection.

For my databinding implementation, I'll use a ViewModel class for the data-source (shown in **Listing 3**). It's a very simple view model; with just a few members suitable for the databinding in the XAML file. It contains a property named **CityWeatherList** which I'll use as the data-source for the TabbedPage ItemsSource property. As you can see in the following snippet, the property is typed as a **ObservableCollection<CityWeather>**.

```
public ObservableCollection<CityWeather>
                                CityWeatherList {}
```

The CityWeather class is a simple data transfer object.

```
public class CityWeather {

  public string CityName { get; set; }
  public string HighTemp { get; set; }
  public string Forecast { get; set; }
}
```

I'll use the ViewModel in the TabbedPage by setting the ItemsSource property In the TabbedPage constructor.

```
public partial class WeatherTabbedBound :
                                TabbedPage {
    public WeatherTabbedBound() {
      InitializeComponent();

      var vm = new WeatherViewModel();
      this.ItemsSource = vm.CityWeatherList;
    }
  }
```

In the XAML for the TabbedPage, I'll create a DataTemplate for each bound item. Remember that the TabbedPage derives from MultiPage.

```
public class TabbedPage : MultiPage<Page>
```

Because of this, I need to use a page class in the data template. In this example, I use a ContentPage and set its Title property via a binding. The results are visible in **Figure 11**.

```
<TabbedPage.ItemTemplate>
  <DataTemplate>
    <ContentPage Title="{Binding CityName}">
```

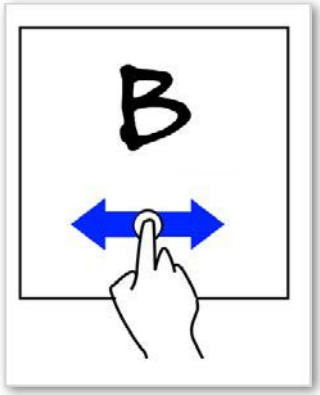


Figure 7: Swipe gesture in Carousel

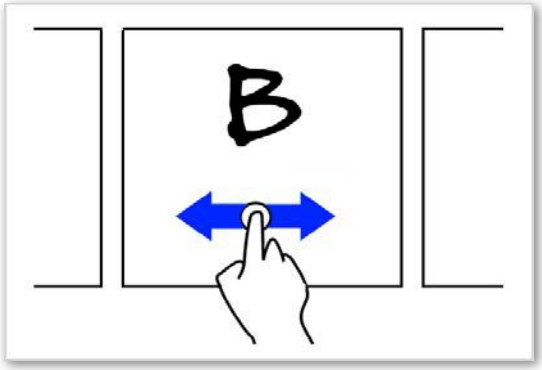


Figure 8: Telltale edges of side pages show while viewing the current page.

Figure 9: The CarouselPage on an Android phone

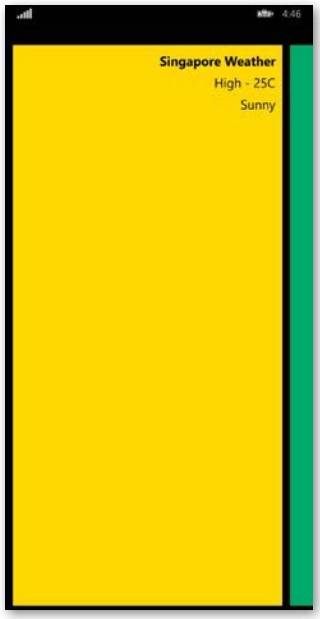


Figure 10: The CarouselPage on a Windows Phone

Figure 11: Binding Page Title in DataTemplate

Figure 12: Binding control properties in DataTemplate

Listing 3: WeatherViewmodel.cs

```
internal class WeatherViewModel : INotifyPropertyChanged {

  public WeatherViewModel() {
    CityWeatherList = WeatherDataSource.GetAllCities();
    this.SelectedItem = CityWeatherList.First();
  }

  public event PropertyChangedEventHandler PropertyChanged;

  private CityWeather _selectedCity;

  public CityWeather SelectedItem {
    get { return _selectedCity; }
    set {SetProperty(ref _selectedCity, value);
    }
  }

  private List<CityWeather> _cityWeatherList;

  public List<CityWeather> CityWeatherList {
    get { return _cityWeatherList; }
    private set { SetProperty(ref _cityWeatherList, value); }
  }

  // A ShowMaster property would be nice in the ViewModel
  // Xamarin binding didn't work with IsPresented property
  // So I removed it from the class

  protected void SetProperty<T>(ref T field, T value,
    [CallerMemberName] string name = "") {
    if (!EqualityComparer<T>.Default.Equals(field, value))
    {
      field = value;
      var handler = PropertyChanged;
      if (handler != null)
      {
        handler(this, new PropertyChangedEventArgs(name));
      }
    }
  }
}
```

```
    </ContentPage>
  </DataTemplate>
</TabbedPage.ItemTemplate >
```

If you're a data binding fan, you'll be happy to learn that the TabbedPage supports binding.

Next, I'll add some child elements to the ContentPage and bind them to the same ViewModel.

```
<DataTemplate>
  <ContentPage Title="{Binding CityName}">
    <StackLayout Padding="10"
                 BackgroundColor="#FF00757D">
      <Label Text="{Binding HighTemp,
              StringFormat=›High: {0}›}" />
      <Label Text="{Binding Forecast,
              StringFormat=›Forecast: {0}›}" />
    </StackLayout>
  </ContentPage>
</DataTemplate>
```

And the results are shown in **Figure 12**.

The full power of DataTemplates are available in the TabbedPage. You can build a complex child page using the any of the built-in views and bind any of the properties of the views to the data-source. The CarouselPage also supports binding and data templates. The only significant difference in the CarouselPage is that the data template must be a ContentPage. That's because of the way the class derives from MultiPage.

```
public class CarouselPage :
             MultiPage<ContentPage>
```

Using a Navigation Bar

When you need more control over the navigation process, perhaps the ability to add or remove pages from the navigation history, turn to the NavigationPage. It exposes the Navigation property, which lets you push and pop items to the NavigationStack. As an added benefit, it adds an appbar to the Android and iOS pages. The user can interact with the appbar to travel forward or backward through the page history. Look for details in the next issue.

John V. Petersen

All Aboard: Maximize Mobile Engagement with Interactive Onboarding

Considering the increasing number of mobile applications on the market, simply getting a user to download your app presents a challenge in and of itself. However, that challenge only represents half of the battle. Once a user downloads the app, you have several other problems to overcome. For instance, how do you engage the user in the short-term and retain them long-term?

Jason Bender

Jason.bender@rocksaucestudios.com
www.controlappdelete.com
www.twitter.com/TheCodeBender

Jason Bender is the Director of Development for Rocksauce Studios, an Austin, Texas-based Mobile Design and Development Company. He has 11 years coding experience in multiple languages including Objective-C, Java, PHP, HTML, CSS, and JavaScript. With a primary focus on iOS, Jason has developed dozens of applications including a #1 ranking reference app in the US. Jason was also called upon by Rainn Wilson to build an iOS application for his wildly popular book and Youtube channel: SoulPancake.

A download doesn't mean much if you can't convert it into a user. Furthermore, users lose significant value unless you can keep them engaged and active.

Think about the application from the user's perspective. They download an app, open it up, and get presented with an initial login screen. At this point, the user hasn't seen your application in action, experienced it first hand, or really had any chance to discover its brilliant allure. What incentive do they have to register? In fact, Apps that require registration before use can lose up to 56% of their users at this point. An initial onboarding experience when an application is launched can play a key role in gripping your potential user and getting them excited to use the app.

In cases where an application has no login, users can get inundated with flows and views that they don't yet know how to interact with. Tapping around in an app without direction can prove frustrating to less experienced users, especially if the app contains unfamiliar user experiences or interactions. In a case like this, progressive onboarding can help guide users as they explore certain aspects of the app, without having to force-feed them all the information up front. Users can get the direction they need when encountering specific flows for the first time.

Developers go to such lengths to create incredibly robust, entertaining, and meaningful experiences in the native space. Why would a developer spend so much time, money, and effort to bring an application's vision to life and not take the little bit of extra time to properly onboard users, guide them through the application's intended use cases, and help them get the maximum value from the product? In this article, you'll take a look at the various onboarding approaches in detail and then code an interactive parallax style pre-login onboarding flow in Objective-C.

What is Onboarding?

Mobile onboarding can take on several forms. Often it refers to the process of guiding a user through using an application for the first time or helping spotlight the app's key features. You only get one chance to make that first impression, and onboarding walkthroughs present great opportunities to showcase the highlights of your product and set user's expectations.

Is Onboarding Really Necessary?

According to Optimizely (a customer experience optimization software company), 80% of downloaded applications only get opened once before they're eventually deleted (https://blog.optimizely.com/2014/05/09/the-optimized-app-ab-testing-goes-mobile/). Converting each app download into a user and retaining each of those users is a difficult challenge for developers. Is onboarding a necessary variable in the equation that solves this problem? If you research mobile onboarding, you will find differing opinions on this question's answer.

> Mobile onboarding is the process of walking a user through an app's features and use cases to help demonstrate its benefits and also setting the user's expectations.

One opinion describes onboarding as a frustrating obstacle that, if needed, signals a fundamental failure in design. It further reasons that needing to walk a user through an application's features indicates an over complicated user experience. These statements may logically have merit, but I feel that the conclusion is too much of a sweeping generalization. It's undeniable that the mobile space has pioneered its own set of rules and practices that users have grown accustomed to; for example, a thumbs up means "like," a star means "favorite," and a trash can means "delete." These straightforward interactions don't need explaining and doing so would likely just annoy your users.

Every app has some level of uniqueness and often developers attempt to push the envelope with new and innovative ideas and concepts. This feeds into the second opinion that onboarding provides a sure-fire way to account for this variation, with user-friendly and intuitive ways to engage and better retain users. Based on my experiences in the mobile space, I feel this second opinion holds the most merit. Let's take a look at some different approaches that help demonstrate the effectiveness of using onboarding.

Pre-registration Onboarding

As mentioned previously, login and signup flows can be a barrier of entry for a potential user. Often, users want to

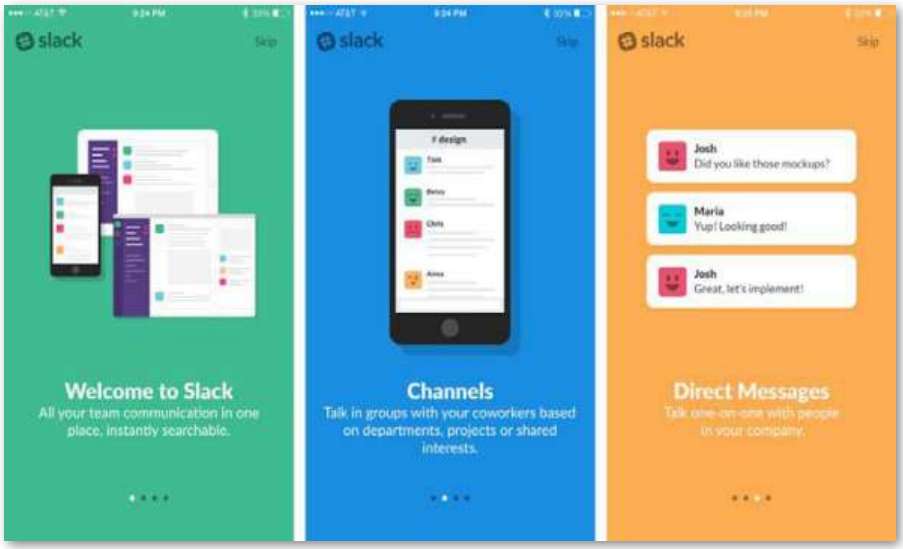


Figure 1: Intro walkthrough flows for a chat application called *Slack*

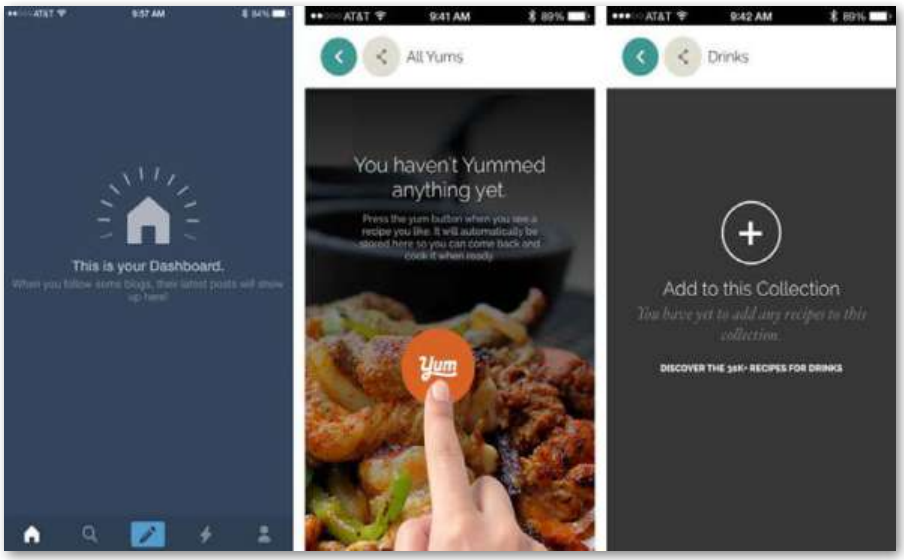


Figure 2: Empty state screenshots taken from Tumblr and Yummly

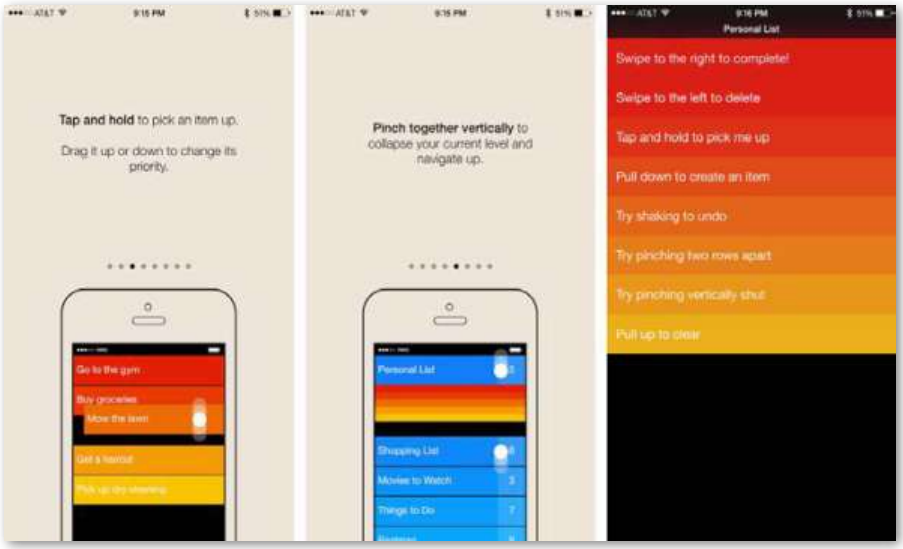


Figure 3: Screenshots from *Clear – Tasks, Reminders & To-Do Lists* created by Realmac Software.

evaluate the usefulness of the application before handing over personal information. Using an interactive onboard flow, similar to the one you'll code later in this article, you can showcase the benefits and most compelling features of your application in an attempt to coax the user through that next step.

Figure 1 demonstrates this concept. The application is called *Slack* (created by Slack Technologies, Inc.) and by looking at the intro slides, you can easily derive that it's a company-based chat program for both mobile devices and the Web. Notice that each screen highlights one specific feature in a clear and concise manner. You want to avoid overdoing it by inundating the user with excess information. Target the most pivotal selling points of the application and, for best results, keep the number of overall slides to a handful. Making the user swipe through ten slides before a registration flow is likely to frustrate them more than inform them.

You'll also notice a skip button in the upper right hand corner. This gives seasoned users the option to jump right in and get started. However, the decision to have a skip option should really depend on how crucial the information is that you're presenting. You will find that users often press a skip button just because it's available, which can cause them to miss important information.

Empty State Onboarding

Often times, an application may rely on user input or user-generated content to populate various views. When a user first accesses the application, those views will most likely be empty. When an application launches to an empty view, the user can be left feeling unsure of what to do next. Empty state onboarding can curb this and nudge the user along the right path.

Take **Figure 2,** for example. The leftmost image is taken from the official *Tumblr* app created by Tumblr. For a new user, when they sign into the application for the first time, their dashboard starts off empty. Imagine that same screenshot with no message in the middle. Users would launch into a blank blue screen with no indication of what that screen does. It would almost seem broken, like something should have loaded but didn't. Instead users can see exactly what this screen does and get a nudge to go follow blogs so content will start appearing.

Along those same lines, the two screenshots to the right (taken from *Yummly Recipes & Grocery Shopping List* created by Yummly) demonstrate similar empty state onboarding techniques. Both clearly describe the action a user must do in order to populate the view. Rather than just saying "no collections added yet," it has a call to action that compels you to go look through drink recipes and start adding them.

Onboarding Unconventional Interactions

Applications that try to innovate using custom interactions that don't necessarily follow the standard mobile conventions make another strong case for onboarding. Take *Clear – Tasks, Reminders & To-Do Lists,* created by Realmac Software, shown in **Figure 3**, for instance. The application uses a minimal user-interface but relies heavily on some unconventional gestures. For instance, to move between varying levels of the navigation hierar-

chy, you pinch in or out as demonstrated by the middle screen in **Figure 3.** It could be argued that this gesture is anything but intuitive. If the application didn't explain this gesture ahead of time, many users may have used the application without ever realizing that functionality existed.

Similarly, the far-left screenshot demonstrates another gesture that allows a user to reorder their tasks. These represent two of the many gestures used throughout the application. Rather than add an additional dozen introduction slides to explain each gesture individually, only the most important gestures get explained up front. However, the developers didn't stop there. The far right screenshot in **Figure 3** demonstrates what your task queue looks like after you make it past the introduction walkthrough. Notice that each task name has a specific direction aimed at introducing you to the remaining gestures. This clever spin on empty-state onboarding transforms a user's initial task list into a tool, helping get them acquainted with the application through use rather than reading about it on another introduction slide.

As intuitive as *Clear's* interactions became once I finished onboarding, without onboarding, the application would be very confusing initially. I likely would get frustrated and opt for another, simpler, to-do list application or I'd poke around until I stumbled my way through it enough to figure it out. What percentage of your user base will take the latter approach over the former? When dealing with interactions or gestures that users don't have exposure to on a regular basis, it's better not to leave that answer to chance and, instead, use onboarding to set your users on the right path.

Progressive Onboarding

In addition to the previously mentioned methods, you can gradually onboard the user when they perform certain actions for the first time, which is called "progressive onboarding." The primary concept here involves recognizing when a user enters an area or flow that they haven't yet seen. At this point, the application uses popovers or overlays to draw the user's attention to key features or actions to help them get acquainted with the unfamiliar flow. The app keeps a record that these flows have been visited so that subsequent visits don't continue to trigger the onboarding overlays.

Figure 4 demonstrates one way to use progressive onboarding. Notice a dimming overlay that helps take focus away from the contents and emphasizes the blue coaching mark on screen. Rather than highlighting all the elements and actions on the screen at once, the coach marks appear in three distinct articles. They appeared over the course of the first five minutes I spent browsing around in the app and gradually directed my attention to specific features and functionality, while still giving me the breathing room needed to explore the application on my own.

Progressive onboarding also works great when dealing with complex workflows that need context to be better understood. For instance, consider an application that helps a user file their taxes. It probably doesn't make sense to instruct a user that they may need a 1099-INT form to show interest income before they've

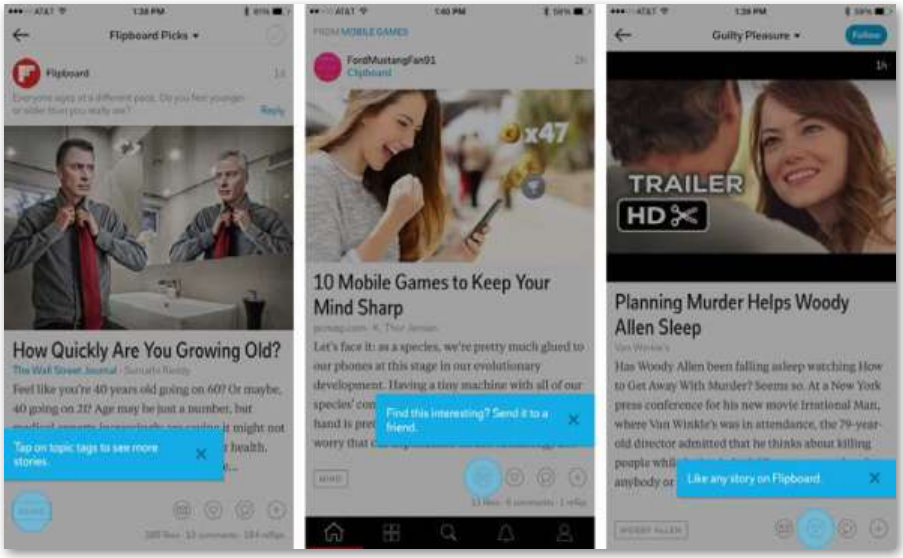


Figure 4: Screenshots from the *Flipboard: Your Social News Magazine* application (created by Flipboard Inc.)

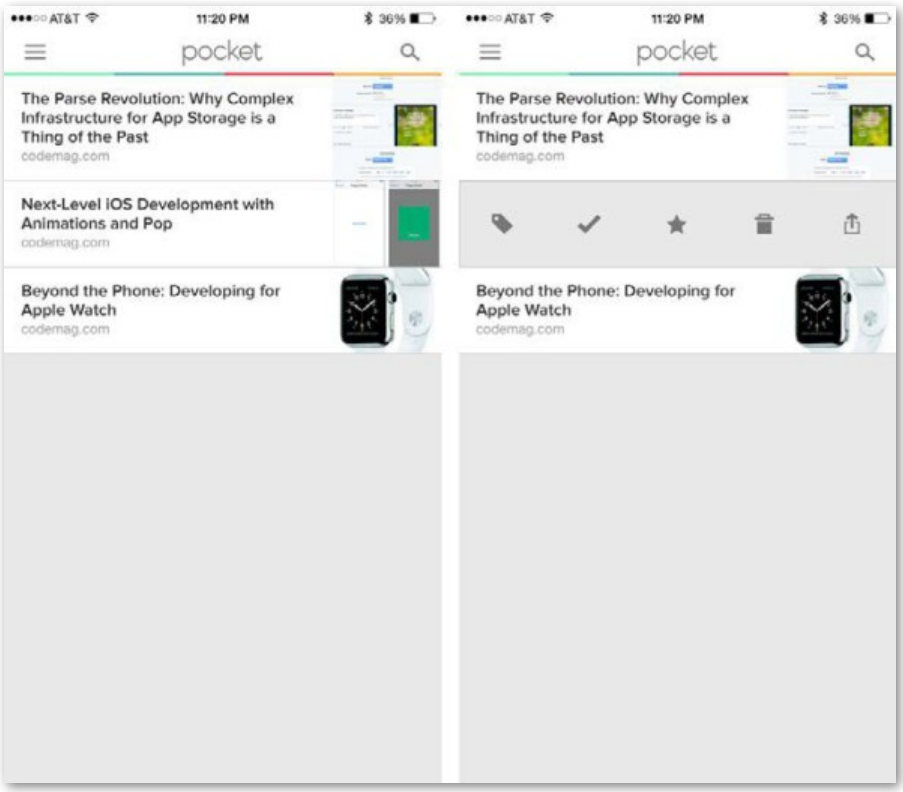


Figure 5: Screenshots from the *Pocket* iOS application

even had a chance to create an account. However, once they've arrived at this portion of the process and indicated that they have interest-bearing accounts, the context of the situation now warrants this explanation. Contextually aware hints and walkthroughs can prove invaluable in helping your users get the individualized help they need based on the use case they currently face.

Additionally, progressive onboarding is useful for spotlighting hidden functionality. If it isn't inherently evi-

dent where to go to access certain features within your application, a simple onboard flow or modal works well to point the user's attention in that direction. **Figure 5** depicts an application called *Pocket*, created by Read It Later, Inc. Notice that on the left-hand side, I've saved several CODE Magazine articles to read at a later point in time. The right hand screenshot shows what happens when you swipe left on any one of those rows. You get a series of additional options to share, delete, favorite, etc. This hidden functionality never gets spotlighted, so unless you discover it via trial and error, it's plausible that you'll never know that it existed. When I saved my first article for later, the application could have taken that opportunity to let me know about the extended options available.

Onboarding Best Practices

The previous sections demonstrate a small sampling of the various shapes and sizes an onboarding process can take. Each has its own benefits and use-cases and which one you choose will be dictated by the functionality of the application you're developing. However, regardless of the onboarding approach you incorporate into your specific application, the following best practices should be taken into consideration:

- **Be Consistent:** Whether it's intro slides or a functionality spotlight, being consistent in both visuals and verbiage will go a long way in creating a coherent experience.
- **Priority is Key:** Your users don't have the attention span to read dozens of slides or modals. Prioritize the information you need to communicate in order for your users to get maximum value.
- **Brief is Better:** When spotlighting or drawing attention to a feature, use high-impact short phrases to communicate the objective rather than long-winded explanations. Users likely won't take the time to read more than a dozen words.
- **Ignore the Obvious:** Avoid reiterating functionality that most users will already understand how to use.
- **Listen to Your Users:** if you haven't done an adequate job onboarding your users, chances are that they'll let you know in some form or fashion. User feedback and reviews can indicate onboarding missteps. Once you recognize a feedback pattern, you can use A/B testing to help better determine which onboarding approach will work most effectively in your application
- **Spotlight Value:** Onboarding isn't just about how to use specific features within an app, it also gives you a chance to communicate the value that your application has to offer and even recommend use-cases that a user would not normally consider.

Onboarding Pitfalls

Onboarding can help make the difference between a good application and a great one; however, onboarding missteps can have the opposite effect, often frustrating users rather than informing them. The following onboarding practices should be avoided:

- **Overdoing It:** The biggest onboarding pitfall is forcing the user to digest more information than is needed. You must find a balance between the minimum information the user needs to use the app successfully and how to show that information in a non-disruptive way. When doing pre-login walkthroughs or spotlights, you'll want to limit the information presented to the core features and highlights of the application. You may have 10 slides of content you want to show, but your user likely won't take the time to digest that much information before using the application.
- **Supplementing Poor Design:** Walkthroughs don't make up for poor design. If you have a confusing or poorly constructed user experience, adding a walkthrough to offset design pitfalls equates to the age-old adage "putting lipstick on a pig."
- **Sensory Overload:** Resist the urge to explain every feature on screen at the same time. Remember the example from **Figure 4**. While the onboarding flows did eventually highlight all the actionable items within that article view, it did so over time as I visited the view repeatedly. This helped to avoid me having to digest five popovers on screen at once.
- **Highlighting the Obvious:** A user likely doesn't need to be told that a heart means "like" or that a star means "favorite." Highlighting these features can annoy users and detract from the apps overall experience.

> Adding a walkthrough to offset design pitfalls is much like "putting lipstick on a pig."

Coding an Interactive Onboard Flow

Now that you have been through some basic onboarding approaches, let's code an interactive onboard flow. In the following example, you'll code a three-page spotlight introduction flow with a background parallax effect. It mimics the Slack example from **Figure 1**. Users will swipe left/right to navigate from slide to slide. Pagination dots indicate on which page they currently reside. When swiping between pages, the background moves at a varying pace to create an illusion of depth.

The walkthrough spotlights an imaginary app called Explore.It, created specifically for this example. **Figure 6** demonstrates what the end result of the slides looks like. The flow consists of several different components. The main container is a UIScrollView with paging enabled. This allows the user to swipe back and forth between the sections. The background image is a UIImageView that sits behind the scroll view and is animated independently. When the page swipe occurs, the background animation gets initiated in parallel.

Getting Started

You want to start with a new project in Xcode. When choosing your project template, select the single-page application template. For this example, I've created a controller called OnboardController to house the onboard flow. Once you have a controller in place, you need to initialize a few variables in OnboardController.m as shown in the following code snippet.

Onboarding Inspiration

Check out the following links for examples of how other applications handle onboarding.

http://pttrns.com/?scid=26

http://pttrns.com/?scid=6

http://uxarchive.com/tasks/onboarding

```
@interface OnboardController (){
    IBOutlet UIScrollView* onboardScrollView;
    IBOutlet UIPageControl* onboardPageControl;
    IBOutlet UIImageView* onboardBackground;

    IBOutlet UIView* slide1;
    IBOutlet UIView* slide2;
    IBOutlet UIView* slide3;
}
```

Notice the elements declared: a UIScrollView to house pages, a UIPageControl to indicate the current page, a UIImageView for the background, and three UIViews, which make up the pages. The next step involves initializing the scroll view and configuring the pages that reside in it. The following function indicates the configuration needed for this implementation.

```
- (void)configureScrollView {
    [onboardScrollView setPagingEnabled:TRUE];
    [onboardScrollView setDelegate:self];
    [onboardScrollView
    setShowsHorizontalScrollIndicator:FALSE];

    [slide1 setBackgroundColor:
    [UIColor clearColor]];
    [onboardScrollView addSubview:slide1];

    [slide2 setBackgroundColor:
    [UIColor clearColor]];
    [onboardScrollView addSubview:slide2];

    [slide3 setBackgroundColor:
    [UIColor clearColor]];
    [onboardScrollView addSubview:slide3];
}
```

The code from the previous function starts by enabling paging on the scroll view. This prevents the scroll view from scrolling continuously. Instead it only scrolls for a duration equal to its width and then automatically pauses. This creates the desired pagination effect. Next, you set the scroll view's delegate to **self**. This will come in handy later when you need access to the scrollViewDidScroll method. In order for that to work properly, you'll want to open up OnboardController.h and add the UIScrollViewDelegate reference. Next, you disable the horizontal scroll indicator because you'll use a UIPageControl to indicate position and a scroll indicator is redundant. Lastly, set all three slides to have a transparent background so that the main background image shows through, and then add them to the scroll view like this"

```
@interface OnboardController : UIViewController
<UIScrollViewDelegate>
```

In Interface Builder, using Auto Layout, the scroll view and the background image gets set to 100% of the screen's bounds. The actual image used in the background is quite a bit larger than the screen width. This allows you to animate it across the screen to create the parallax effect. Additionally, the slides each get created as their own UIView. For the purpose of this illustration, a single image gets used for the slide contents. That image should be centered in its corresponding slide. Lastly, a title and skip button get added on top of the scroll view

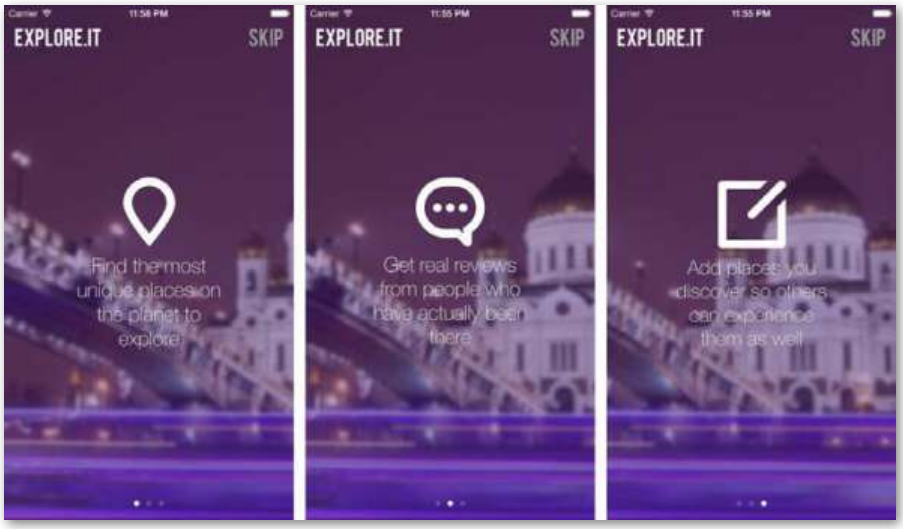


Figure 6: Screenshots of the sample onboard application built in this article

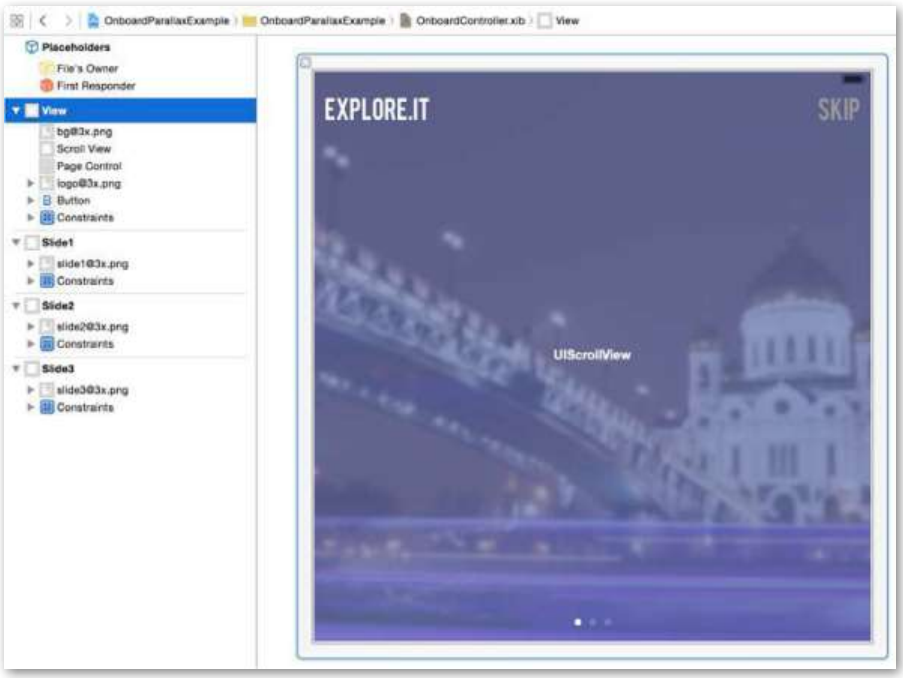


Figure 7: Represents what Interface Builder should like as described in the example

and a UIPageControl added to the bottom. The state of interface builder should resemble **Figure 7**.

The visuals can take any form you like. The important thing to monitor is the structure, which is a UIScrollView with paging enabled and three slides, in the form of UIViews, and added as subviews. In order for this to paginate properly, you need to set the frame of the slides to equal the screen width. You also need to set the content size of the scroll view to be three pages wide, which in this case equates to the screen width times three. You can do all of this in the viewDidLayoutSubviews method, as shown in the following snippet.

```
- (void)viewDidLayoutSubviews {
    [onboardScrollView
    setContentSize:CGSizeMake(SCREENWIDTH * 3,
```

Beyond Mobile

Onboarding doesn't only apply to mobile applications. It's equally important on the Web. The following link breaks down the onboarding flows of several popular Web apps.

https://www.useronboard.com

```
    onboardScrollView.frame.size.height)];

    [slide1 setFrame:CGRectMake(0, 0,
    SCREENWIDTH, SCREENHEIGHT)];
    [slide2 setFrame:CGRectMake(SCREENWIDTH, 0,
    SCREENWIDTH, SCREENHEIGHT)];
    [slide3 setFrame:CGRectMake(
    (SCREENWIDTH * 2), 0, SCREENWIDTH,
    SCREENHEIGHT)];
}
```

SCREENWIDTH and SCREENHEIGHT represent constants, declared as follows.

```
#define SCREENHEIGHT  (int)[[UIScreen mainScreen]
                           bounds].size.height

#define SCREENWIDTH  (int)[[UIScreen mainScreen]
                           bounds].size.width
```

Notice that the content size of the scroll view gets set to three times the screen width and since the view's width equals the screen width, this equates to three full pages to swipe among. Next, each slide gets positioned. Notice that the first slide gets positioned at an x-coordinate equal to 0. The next slide gets set with an x-coordinate equal to SCREENWIDTH. Since each page is the width of the screen, you want to align the left edge of the second page just off of the right edge of the screen so it can swipe in when the user moves from page 1 to page 2. Lastly, slide 3 gets set to SCREENWIDTH * 2.

Connecting the Dots

At this point, you should have three slides that you can move back and forth among. Next up, let's add in the UIPageControl to indicate which page is currently visible. To do this, add the following code to the scrollViewDidScroll delegate method. Each time the user swipes left/right, this code detects and visually indicates the page the user lands on.

```
- (void)scrollViewDidScroll:(UIScrollView*)sender{
    // Update the page when more than 50% of the
    // previous/next page is visible
    CGFloat pageWidth =
      onboardScrollView.frame.size.width;

    int page =
      floor((onboardScrollView.contentOffset.x -
      pageWidth / 2) / pageWidth) + 1;

    onboardPageControl.currentPage = page;
}
```

The last piece of the puzzle involves animating the background image when the page changes. To do this, add the following method to your controller.

```
- (void)pageChanged:(int)page {
    [UIView animateWithDuration:.5 delay:0.0
      options: UIViewAnimationOptionCurveLinear
      animations:^{
        [onboardBackground setFrame:
          CGRectMake(0 + (page * -80), 0,
          onboardBackground.frame.size.width,
          onboardBackground.frame.size.height)];
      }completion:^(BOOL finished){}];
}
```

Once added, call this function from the end of the scrollViewDidScroll function that you edited earlier, like so.

```
[self pageChanged:page];
```

From the code, you can see that each time this function triggers, it animates the background image on the x-axis relative to the current page. If the user lands on the first page (page = 0), the image returns to its initial state. Otherwise, the image moves 80 pixels left/right each time the slide changes. The distance this image travels directly influences the prevalence of the parallax effect. In this setup, the slide must move the entire width of the screen (it varies based on individual devices but is at least 340px) while the background image only shifts by 80px. This difference in space simulates distance between the elements in the foreground and the scene in the background.

Wrapping Up

When taking the time to design and develop mobile applications, consider onboarding. With as little effort as it takes to implement (this example is about 20 lines of code), an onboarding flow can add that extra engagement needed to draw your users in and help them get the maximum benefit that your application has to offer. The code covered in this example has been included with this article so you can test it out for yourself. Additionally, check the sidebar for some links to various onboarding examples for inspiration.

Jason Bender

CODE

Qualified, Professional Staffing When You Need It

CODE Staffing provides professional software developers to augment your development team, using your technological requirements and goals. Whether on-site or remote, we match our extensive network of developers to your specific requirements. With CODE Staffing, you not only get the resources you need, you also gain a direct pipeline to our entire team of CODE experts for questions and advice. Trust our proven vetting process, and let us put our CODE Developer Network to work, for you!

Contact CODE Staffing today for your free Needs Analysis.

Helping Companies Build Better Software Since 1993

www.codemag.com/staffing
832-717-4445 ext. 9 • info@codemag.com

CODE STAFFING

WTF.js

The last ten years have been interesting, to put it mildly. Ten years ago, some of us bought desktops, some of us bought laptops, and those of us with loads of cash bought both. Since then, you have a desktop for heavy-duty work, a laptop for portable stuff, and you may even have both a Mac and a PC. You also have a tablet for airplanes and cafes, a smartphone for everywhere else,

Sahil Malik
www.winsmarts.com
@sahilmalik

Sahil Malik is a Microsoft MVP, INETA speaker, a .NET author, consultant, and trainer.

Sahil loves interacting with fellow geeks in real time. His talks and trainings are full of humor and practical nuggets. You can find more about his training at http://www.winsmarts.com/training.aspx.

His areas of expertise are cross platform Mobile app development, Microsoft anything, and security and identity.

and a smartwatch for even more everywhere else(!?). Some of us even have 17" screens in their new fancy cars. There are too many platforms—it's overwhelming. Not only is my wallet getting tired, this universal connectivity creates a real challenge for developers and for the people paying for those developers. I don't even want to begin thinking of the challenge it's creating for recruiters.

Ten years ago, if I wanted to build a product, I wrote a website or maybe a WinForms app. Today, I have to write code to cover every platform. Like I said, it's getting overwhelming. But there is this one language that runs everywhere: JavaScript. Open Web technologies, such as HTML and JavaScript, may have their shortcomings, but given their wide applicability, they are being used everywhere.

The second major megatrend powering this JavaScript revolution revolves around performance. I feel mildly amused when people brag about the huge amounts of storage they can now stuff into a small SD card. Yes, that's very impressive, but I wonder why people don't talk as much about how far JavaScript engines have come in the last few years. It wasn't that long ago that we were dealing with IE6. The debugging tools sucked and the performance was incredibly bad. Compared to the new Edge browser on the same hardware, IE6 gives you a million times slower performance. In other words, JavaScript engines have improved a thousand times more than your average storage technology.

The third megatrend is the widespread adoption of AJAX, and the amazing platforms that let us treat a Web page as a canvas for applications. Platforms such as AngularJS and React make it almost too easy to be free of the issues of postbacks, introducing design patterns such as MVC, and even converting to native code for mobile.

Naturally, this has had major implications. We're writing a LOT of JavaScript lately and this trend is not going to change anytime soon.

The Problem with JavaScript

As amazing as JavaScript is, the language itself plain sucks. It's amazing because it runs everywhere, but it sucks because it's so easy to write bad code in JavaScript, especially if you're coming from a C-like language such as Java, C++, or C#. In this article, I'll point out some of the most common mistakes that new-to-JavaScript developers make.

Why does JavaScript suck? The answer is simple: history. JavaScript was created by a really smart guy called Brendan Eich over a period of a week or so. He did it because Microsoft was largely perceived as evil, and his aim was to create a programming platform in the Netscape navigator. That way, the browser would run everywhere, and the OS wouldn't matter as much. HAHA! Down with Microsoft. You can see how well that worked out.

There was another company in Silicon Valley called Sun Microsystems, and they had their own plans to bring down the evil empire of Microsoft. They created a language called Java. Write once, run everywhere, or so they claimed. If you can run your code everywhere, why do you need a single OS? HAHA! Down with Microsoft. You can see how well that worked out for them, too.

The two Microsoft adversaries decided to shake hands. Then came the interesting challenge of having two tools solving the same problem. One was a browser scripting language and the other was Java. Clearly, this was a problem they could solve, not using technology but with clever marketing. Unsure of what to do, they decided to call this new scripting language "JavaScript," the simple-minded younger brother of Java. The connotation was that you'd use Java for *real* work, and JavaScript for quick-and-dirty browser stuff. This unfortunate choice has been confusing recruiters and managers ever since.

Meanwhile, 1500 miles away in Redmond, Microsoft took note. They decided to get serious about both Java and JavaScript. For legal reasons, they couldn't just make Java and JavaScript available on their OS and browser. So they studied these languages very diligently and created Visual J++ and JScript, the Microsoft implementations of Java and JavaScript. Visual J++ had lots of Microsoft nuances thrown into it. But JScript was a very good copy of JavaScript. They copied all the mistakes and problems of the language too—with great accuracy. I wish they were as diligent in creating Windows ME, Vista, and 8. Wishes aside, they introduced an Internet Explorer-only language called VBScript that offered much richer functionality than JScript on the Windows platform. Even though IE and Windows had a footprint of 95% or more, open Web technologies eventually killed VBScript. The new language was called "Jscript." As good a copy as it was of JavaScript, they couldn't legally call it JavaScript because Sun Microsystems was so litigious that they would sue a coffee shop with the word "Java" in its name. Now look who's evil.

As time went on, Sun Microsystems went on corporate welfare along with Microsoft, the guard changed, and the world in general decided that it would be good to standardize the language. So they submitted it to a standards body, agreed on the standards, but needed a good name for the language. They couldn't come up with a good name, so they called it "ECMAScript" (for now!), and maybe they would come up with a better name next week. That renaming never happened.

So now we're stuck with the name ECMAScript and even more confused recruiters (and managers).

The crux of this story is that the language, created with loads of imperfections, has grown, been standardized, and has been implemented all the way from your Pebble watch, to a Tesla screen, to Smartphones, browsers, desktops, and servers. This language, originally intended for 100-line quick–and-dirty scripts, is now being used to write enterprise applications with millions of l ines of code.

Take all of that, add in the issues in the language, the lack of good tools, and the blight of overconfident developers, and you have way too much bad JavaScript floating around.

Writing good JavaScript requires you to understand three things:

- The most common mistakes JavaScript developers make
- Debugging techniques that will save you a lot of time and hair
- ECMAScript 6 (ES6) and TypeScript, also known as the light at the end of the tunnel

Of course, there's a lot more to learn besides those three bullet points on this topic. In this article, I'll cover the most common mistakes that people make. In subsequent articles, I'll talk about how debugging helps and how ES6 and TypeScript will help.

Common JavaScript Mistakes

By far, the following isn't a comprehensive list of mistakes that smart developers can make. Developers can be rather creative, but here are a few of the most common mistakes.

Polluting the Global Namespace

This is pretty obvious: There is only one global namespace. If I create a variable called **dummy**, and you create a variable called **dummy**, the one who created it last wins. The best idea is: As much as possible, don't create global variables. The problem is that sometimes it's very easy to accidentally create global variables. For instance, check out the code below

```
(function(){
    var notglobal = 1;
    accidentalglobal = 1;
})();
global = 1;
```

In this code, the accidental global—even though it's inside a function scope—ends up being a global variable. This is because you didn't use the **var** keyword. The funny thing is, the above code will work, it's just prone to bugs. Let me explain why. Observe the following snippet for loop.

```
function doSomethingUseful() {
    for (i = 0; i < iteratable.length; i++) {
        iteratable[i];
    }
}
```

That may appear second nature to most C# programmers, but in JavaScript, it's truly awful. The problem is that the variable "i" (is an accidentally global variable. The right way to write the code would be to say "var index = 0" instead of "i = 0". And even **var** doesn't 100% protect you. Using **var** simply means that the variable is restricted to the function scope, but within the function, you may still be overwriting another variable called index. At least you'd be shooting your own foot then, not someone else's, but still, the best practice is that if you really wish to restrict a variable in its own scope, you should put it in its own *function* scope. That can get cumbersome, though, because you get way too many nested functions, so in reality, there's always a practical tradeoff between readability and maintainability. Luckily, ES6 has introduced a new keyword called **let**, which truly restricts your variable to the curly brackets scope.

Semicolon Carelessness

When writing code, always be clear and explicit about your intent. This applies to every language, and as time goes by, the language developers introduce interesting shorthand syntaxes that make code hard to read and easy to confuse. The irony is that they do so in the name of productivity. The few keystrokes they save you aren't worth the hair you will eventually lose figuring out what that code is actually doing. C#, I am looking at you.

Still, in JavaScript, using semicolons is optional. However, you should always end your statements with a semicolon. For instance, observe the code below:

```
function functionThatReturnsSomething() {
    return
    {firstName: „Sahil“,lastName: „Malik“}
}
```

Although it may appear that the code is returning an object, it isn't. It is, in fact, returning a null, and then the free-hanging object either has no effect, or errors out, depending upon the browser. This is because the code got interpreted as "**return;**" instead of **return object**. Which brings me to my next point.

Curly Brace Fights

Statements such as if**, while**, **for**, etc., that require scopes denoted by curly braces can be written in three different ways. The first method is where you place the curly brace right after the statement, as shown below:

```
if (condition ) {
    statement;
}
```

The second, typically done by overconfident developers looking to show off, is where you omit the curly brace completely:

```
if (condition) statement;
```

And the third, usually done by developers who haven't ventured beyond Visual Studio much:

```
if (condition)
{
    statement;
}
```

The first method is the best method. It's error free, it's clear what your intent is, and it won't do accidental semicolon insertions under any condition.

Reserved Keywords

It's a good idea to keep an eye on all the reserved keywords both for the JavaScript you're writing today, and what the browser may upgrade to in the future. Using a future reserved keyword may mean that your application will break when the browser updates. I usually prefix variable names with a short prefix to keep my variable names separate from all reserved keywords.

Using typeof Blindly

The **typeof** keyword is supposed to return the type of the object, for instance, **typeof 1** should return Number, and **typeof "sahil"** should return a string. Unfortunately, that's where the fun ends. Because beyond that, **typeof** returns all sorts of irregular results for objects. For instance, **typeof null** is an object, **typeof []** should be an array, but its object, and **typeof NaN**, which stands for "not a number," is a number!

You can't rely on **typeof** to do type checking for you. You have to write polyfill methods to do this job. For instance, to really check whether or not an object is an array, you have to write a method, as shown in **Listing 1**.

When parseInt isn't Very Reliable

One of the hardest things in JavaScript is that there is no static typing of variables. This means that tools can't help you much, and you tend to make more mistakes confusing data types. You declare a variable using the **var** keyword, which is optional. But when you've declared something, you can easily redefine it to anything you wish. Even if you don't redefine it accidentally, sometimes JavaScript internally changes a number to a string, or a 64bit number to a 32bit number etc.

Frequently, you have a variable that you think is a number, but in order to do math on it properly, you want to convert it into a strongly typed number. For instance, "1" (string) needs to be converted into 1 (Number). For situations like these, and more, use a method called parseInt. Unfortunately, parseInt is also not very reliable. What makes it worse is that the behavior can be inconsistent between browsers. For instance, consider this code below:

```
parseInt(„07“);
```

Some JavaScript engines interpret the above as 7 being converted with an octal radix, because you have a leading zero. The return value is 10. The leading zeros can be quite common when parsing dates for instance.

Listing 1: The is_array method

```
var is_array = function (value) {
    return value &&
        typeof value === ‚object‘ &&
        typeof value.length === ‚number‘ &&
        typeof value.splice === ‚function‘ &&
        !(value.propertyIsEnumerable(‚length‘));
}
```

As a defensive coding practice, always use parseInt like this:

```
parseInt(„07“, 10);
```

The second parameter is the radix – always specify the radix explicitly.

But it doesn't end there. Consider these lines of code:

```
parseInt(„12“);
parseInt(„12 monkeys“);
parseInt(„dozen monkeys“);
parseInt(„monkeys that are 12“);
```

The first two lines return 12, and the last two return NaN, not a number. The parseInt looks at the first few number-like characters and does its best in returning what it could understand. This behavior is also different from most other languages.

Number Craziness

JavaScript has only one datatype for numbers, and it's called **Number**. It's a 64bit number most of the time. When you do a bitwise operation, the number internally converts to 32bit, and you could lose some fidelity. Also, the **Number** datatype is guaranteed safe for only 15 digits. To try this, run the next line of code (note that there are sixteen 9s).

```
console.log(9999999999999999);
```

You'd be shocked to see that the above line of code outputs 10000000000000000. It rounded it up for me. I didn't ask for it, but it did it anyway.

It's for these purposes that **Number** includes properties called MAX_SAFE_INTEGER and MIN_SAFE_INTEGER. If precision is important to you, you should stay within those boundaries.

But your headaches don't end there. Try the following in any browser's console:

```
0.1 + 0.2
```

Even a three-year-old could do the above math properly. JavaScript is almost 30 years old and can't do this basic math properly! JavaScript prints out the answer to the below as 0.30000000000000004. Technically speaking every language does this. In C#, if you add two floats, you'll get the same result. The big difference here is that languages such as C# and Java allow you to create numbers as decimals, making it very easy to do such calculations accurately. Those of us who have written financial programs know about the salami slice problem (see the sidebar) far too well. JavaScript requires you to do some interesting math, using the Math object to achieve the same results.

NaN in JavaScript stands for "not a number." Unfortunately, **typeof NaN** returns "number." How is "not a number" a number?

What's more interesting is if you compare **NaN == NaN**, the result is false. But if you set **x = NaN**, and check **x !==**

x, it evaluates to true! These operators in JavaScript also cannot be trusted!

Operator Issues

JavaScript has many operators, but the two you need to be especially careful of are addition/subtraction operators and equality check operators.

For instance, consider the code:

```
var i = 1;
i = i + „“; // oops!
i + 1;  // evaluates to the String ‚11‘
i - 1;  // evaluates to the Number 0
```

As you can see, the **+** operator is used to concatenate both strings and to add numbers. This would otherwise be fine, but when paired with a lack of static typing, data type deduction, or, should I say, data type guess work, this can be quite problematic. You have similar issues when using the **++** operator as well. For instance, consider the next bit of code:

```
var j = „1“;
j++; // j becomes 2
var k = „1“;
k += 1; // k becomes „11“
```

As can be seen, the **++**, **--**, **+=** and **-=** operators work in different ways. I prefer to avoid the **++** operator because it's confusing and doesn't add any value. Just keep your code clear by being explicit about what you mean. That way, there will be fewer errors, and it'll be easier to understand for the next unfortunate guy.

As if the **+** operator madness weren't enough, JavaScript plays interesting tricks when checking for equality. For instance, check out the code in **Listing 2**.

All of these statements evaluate to true. How could that be possible? How is **false** not equal to "false"? The reality is that you should just avoid using **==** checks for equality. Instead you should use the typesafe checks using the **===** operator.

The with Statement

JavaScript has an interesting "with" statement that can lead to code behaving differently than you expect, depending upon the data at the time of execution. It almost goes without saying that you should avoid **with**. For instance, check out this code:

```
with (obj) {
    a = b;
}
```

It's impossible to tell what the code does by just looking at it. The code behaves differently if there is an **obj.a** property versus if there is a variable called "a", and if those have any values in them.

Code that changes its execution based on data is by definition unreliable. Long story short: Avoid using **with**.

Arrays

JavaScript provides one way of storing data (besides objects) and that's arrays. Technically speaking, arrays are also objects, but let's leave that pedantic discussion for another day. The issue with arrays in JavaScript is that they're slow. They're slow and JavaScript has no alternate fast way of storing and retrieving data. Secondly, managing arrays is quite slow as well. You can delete an element in an array, but it leaves holes of undefined values in the array. "Packing up" the array and getting rid of those undefined values can be quite expensive.

Listing 2: All these statements evaluate to true

```
0 == „“
0 == „0“
0 == „ \t\r\n „
„0“ == false
null == undefined

„“    != „0“
false != „false“
false != undefined
false != null
```

I wish the problems ended there. Consider the next bit of code:

```
var arr = [1,2,3]
var total = 0;
for (var i in arr) {
    total = total + i;
}
```

Take a guess: What's the value of **total** at the end of the loop? Will it write **6**? No, it won't, because the "i" variable doesn't actually get the value of the array element. Instead, it gets a zero-based indexer. So will it print "3" (0+1+2)? Sadly not even that. It prints **0012**. Even though the indexer is supposed to be a number, the **+** operator interprets it as a string. In order to get the proper output, you have to add the following line in the **for** loop:

```
i = parseInt(i);
```

I was just checking to see if you were awake. That line has a bug in it too. What you really have to do is add the following line:

```
i = parseInt(i, 10);
```

This is because you must specify the radix or risk having your numbers interpreted as octal numbers. I explained this a little earlier in this article.

Now, I wish I could say that this is all that is wrong with JavaScript arrays, but wait until you try to sort them. For instance, consider this code:

```
[40, 100, 1, 5, 25].sort();
```

The output might surprise you! The output looks like this:

```
[1, 100, 25, 40, 5]
```

Upon close observation, you'll notice that the numbers are sorted as strings, even though the inputs were clearly numbers. If you really wanted to sort them as numbers, you have to pass in a comparer function like this:

Salami Slicing

Salami slicing refers to a series of many small actions, often performed by clandestine means. They're often an accumulated whole that produces a much larger action or result that would be difficult or unlawful to perform all at once.

Listing 3: The confusing this keyword

```
var cars = {
    description: „Cars I like“,
    carDetails: [{make:“Tesla“, model:“Model S“}],
    tellMeAboutcars: function() {
        console.log(this);
        this.carDetails.forEach(function(car) {
            console.log(this);
            console.log(car.model + „:“ + this.model);
        });
    }
}

cars.tellMeAboutcars();
```

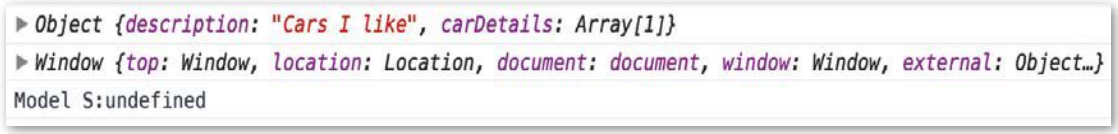


Figure 1: The output of Listing 3.

```
[40, 100, 1, 5, 25].sort(
    function(a,b){return a-b;});
```

This is so confusing! Speaking of “this”...

This is So Confusing

JavaScript has a special keyword called **this**. The value of **this** changes depending upon the context; much like in the English language, “this” can be confusing. For instance, check out the code in **Listing 3**.

When you run this code, it produces the output shown in **Figure 1**.

This is clearly not what you expected. Why is “this.model” undefined? It’s because when you switched into the inner function’s scope, the value of **this** changed to the context on which the function was being called, and that context, the value of **this** at that time, was the window object. It’s therefore quite common practice to store the value of **this** as a private variable in the outer object or function, and reference the value of the object using the outer variable. Outer variables are accessible in inside scopes.

Scopes! How could I not mention scopes and closures in this article?

Understanding Scopes and Closures

Scopes are how you keep your code separate from other code. You want to keep your code separate because you don’t want others to accidentally change your implementation, and vice versa. In most high-level languages, **scope** is defined using curly braces. For instance, observe the code shown in **Listing 4**.

Listing 4: Ineffective scopes using braces

```
{
    var outerScope = 1; {
        var innerScope = 2;
        document.writeln(innerScope);
        document.writeln(outerScope);
    }
    document.writeln(innerScope);
    document.writeln(outerScope);
}
```

The braces in the code shown have no meaning. They might as well not be there. In JavaScript, you don’t have **block** scope. You have **function** scope. If you really wanted to have your own scope, you’d have instead written your code as shown in **Listing 5**.

By separating out your scopes into functions and actually calling the functions, you’ve now created separate scopes where you can “hide” values from others. And note that I’m calling the function; you could make a self-calling function, like this:

```
(function() {
    // write your logic here
})();
```

This sort of a self-calling function is called a closure. It gives you an area that’s all for you. There’s no danger of stepping over anyone else’s variables or polluting the global namespace. This is how you *should* write JavaScript.

Function Hoisting

There are many other JavaScript WTFs to learn. I’ll end with a really weird one called JavaScript hoisting.

Hoisting refers to the phenomenon where JavaScript takes the declaration of all variables and moves them to the top of your code automatically, irrespective of where they may appear in your code. The advantage, of course, is that you can write a function anywhere, and you can call it from anywhere. You don’t have to worry about writing your functions in the right order. Do note, though, that initializations are not hoisted. The values are assigned where you left them; only the declarations are moved to the top.

Although that’s a splendid idea, given how flexible JavaScript can be, some clever developers end up shooting themselves in the foot. For instance, let’s say that I have ten buttons on the page and I want to create a function for handling the **onclick** event for each of those ten buttons.

What should I do to achieve this? Perhaps a **for** loop, and inside the **for** loop, I can write **button.onClick = function() {..}**? WRONG!

Listing 5: The right way of doing scopes

```
var outerFunction = function(){
    var outerScope = 1;
    var innerFunction = function(){
        var innerScope = 2;
        document.writeln(innerScope);
        document.writeln(outerScope);
    };
    document.writeln(outerScope);
    innerFunction();
};
outerFunction();
```

By doing this, you just hoisted ONE declaration of the function, not ten. Remember, JavaScript can't know ahead of time, during hoisting, how many times you might run the loop. As a result, you get ONE function, not TEN.

The proper way of handling this would be to move the creation of the function outside loop and return new instances of the function as many times as the loop runs.

In short, never put function declarations inside **loops** or **if** blocks! It will only lead to problems and more hair loss.

Summary

This article walked you through some common mistakes that the best of us make when working with JavaScript. Yes, JavaScript can suck! But don't forget, its dynamic nature is what also makes it so flexible. And its wide applicability is what makes it so productive and useful. One thing is clear: You need to master JavaScript. The intent of this article was not to scare you away from JavaScript, but point out some of the landmines that you could step on.

And thankfully, these problems are well recognized and a lot of smart people are working on sustainable solutions to these issues. The solutions may be in terms of frameworks, or they may be in terms of improvement to JavaScript itself, or perhaps creating transpilers and new languages such as TypeScript.

In subsequent articles, I'll tackle some of those improvements that make JavaScript a lot more fun.

Until then, happy hacking!

Sahil Malik

LEADTOOLS HTML5 / JavaS

Developing your enterprise-level, cross-platform imaging application has never been easier.

DESKTOP SERVER WEB MOBILE

Sacrifice nothing by developing your application with the award-winning HTML5 and JavaScript technology in LEADTOOLS. Large enterprises and corporations demand the utmost in performance, quality and dependability, and rightly so. Get all of that and more with LEADTOOLS, the world's most powerful and widely adopted imaging SDK in use by nearly every Fortune 1000 company.

OCR ***Barcode*** ***Forms*** ***PDF***

Document Conversion ***Document Cleanup*** ***Viewers***

Annotations ***DICOM*** ***PACS*** ***HL7***

File Formats (150+) ***Compression*** ***Multimedia***

Download a free evaluation today and get started on the path of exceeding client expectations, coming in under budget, programming with less stress, and earning a greater ROI.

.NET Windows API WinRT Linux iOS OS X Android HTML5/JavaScript

cript SDK

WWW.LEADTOOLS.COM

LEADTOOLS® V19
THE WORLD LEADER IN IMAGING SDKs

The Baker's Dozen: 13 Miscellaneous Transact-SQL Tips

There's a line in the movie "Lord of the Rings" about "the ties that bind us". One thing that binds many application developers is the need to write T-SQL code. In this latest installment of The Baker's Dozen, I'll present 13 T-SQL programming tips. Although many of these are rather introductory topics that many SQL developers already know, there should hopefully be a few items in here that are new to intermediate SQL programmers.

Kevin S. Goff
kgoff@kevinsgoff.net
http://www.KevinSGoff.net
@KevinSGoff

Kevin S. Goff is a Microsoft SQL Server MVP, a Database architect/developer/speaker/author, and has been writing for CODE Magazine since 2004. He's a frequent speaker at community events in the Mid-Atlantic region and also speaks regularly for the VS Live/Live 360 Conference brand. He creates custom webcasts on SQL/BI topics on his website.

What's on the Menu?

True to the traditional Baker's Dozen 13-item format, here are the 13 topics I'll cover in this issue:

- Cumulative aggregations before SQL Server 2012
- Cumulative aggregations using SQL 2012
- Moving Averages in SQL 2012
- **Baker's Dozen Spotlight**: calculating elapsed time across rows
- UNPIVOT versus UNION
- The multiple aggregation pattern (and anti-pattern)
- Differences between IN and EXISTS
- A use for RANK functions
- Cursors versus Table Types for performance
- Calculating aggregations across rows
- Parameter sniffing and stored procedure execution plans
- Stored Procedures that try to query for one parameter value or ALL
- The difference between WHERE and HAVING

1: Cumulative Aggregations before SQL Server 2012

Programming challenge: As shown in **Figure 1**, based on Orders in AdventureWorks in 2006, write a query to retrieve the first order (by vendor) that gave the vendor a cumulative order total of at least $10,000.

T-SQL developers often struggled with calculating cumulative aggregations ("running totals") prior to SQL Server 2012. This was a scenario where developers often looked for a row-by-row approach instead of a set-based approach. Even the CROSS APPLY feature in SQL Server 2005 only partly helped with this issue, as developers still needed to write moderately complicated subqueries to conditionally aggregate from the first row of a scope to the most current row.

VendorID	Name	POID	OrderDate	TotalDue	Cumulative
1494	Allenson Cycles	161	2006-07-01	9776.2665	19552.533
1498	Trikes, Inc.	72	2006-02-25	29233.0789	29233.0789
1506	Greenwood Athletic Company	38	2006-02-16	48489.6873	48489.6873
1508	Compete Enterprises, Inc	23	2006-01-15	41234.0639	41234.0639
1526	International Bicycles	45	2006-02-16	31164.2541	31164.2541
1530	Comfort Road Bicycles	22	2006-01-15	31023.8639	31023.8639
1538	Vista Road Bikes	76	2006-03-12	41837.1504	41837.1504
1542	Hill's Bicycle Service	40	2006-02-16	31323.7885	31323.7885
1544	Circuit Cycles	100	2006-04-09	7725.4638	15450.9276
1556	West Junction Cycles	77	2006-03-12	28216.0589	28216.0589

Figure 1: For Orders in 2006 shows the first order (by vendor) that gave the vendor a cumulative total of at least $10,000.

Listing 1 shows one solution for anything between SQL Server 2005 and SQL Server 2008R2, inclusive. The solution involves a CROSS APPLY to query the order table in the outside and then on the inside as a subquery (similar to a SELF JOIN), and to sum the order amounts in 2006 for all orders for that vendor with a Purchase Order ID on the "inside" less than or equal to the Purchase Order ID on the "outside." Yes, this approach assumes that the purchase order IDs are assigned in pure sequential/chronological order. You could also use an Order Date if the Order Date is down to split-second granularity where no two orders for the same vendor could have the exact same time.

This pattern simulates a "perform varying" operation to sum up the order amounts for all orders prior to the current order. Once you've established a cumulative amount as a common table expression (and filter on the rows where the cumulative value exceeds $10,000), you can query into that common table expression and retrieve the first row (with a TOP 1 or a RANK() value of 1) for each vendor. This has never been an attractive or elegant solution, but it certainly works.

Fortunately, Microsoft added functionality to the T-SQL language in SQL Server 2012 to make cumulative aggregations a little easier, and that leads to the next tip.

2: Cumulative Aggregations using SQL 2012

Listing 2 shows a basic example of how to aggregative cumulatively, using the new SQL Server 2012 keywords **ROWS UNBOUNDED PRECEDING**, as shown below. This feature allows developers to sum/aggregate rows from the first row in a group to the "current" row based on a specific order.

```
select PurchaseOrderID, OrderDate, TotalDue,
  SUM(TotalDue) OVER (PARTITION BY VendorID
                      ORDER BY PurchaseOrderID
            ROWS UNBOUNDED PRECEDING ) AS
      CumulativeAmount FROM…..
```

This gives you a cumulative sum of TotalDue for each row, based on all prior orders for the Vendor. But how? There are two related answers. First, the SUM(TotalDue) OVER tells SQL Server that you want to aggregate over a set (or

Listing 1: Cumulative Aggregation (Running SUM) prior to SQL 2012

```
;with CumulativeCTE as
   (
   select POH.VendorID, POH.PurchaseOrderID ,
         cast(POH.OrderDate as date) as OrderDate,
          POH.TotalDue, RunningTotal.Cumulative
   from  Purchasing.PurchaseOrderHeader   POH
       cross apply (   select sum(TotalDue)  as Cumulative
              from Purchasing.PurchaseOrderHeader Inside
                     where Inside.VendorID =poh.VendorID and
                       Inside.PurchaseOrderID <= POH.PurchaseOrderID
                       and year(OrderDate) = 2006
                   ) RunningTotal
                  where year(OrderDate) = 2006
                        and Cumulative > 10000 )

  select   VendorID, Name,  CumulativeCTE.PurchaseOrderID as POID,
           cast(CumulativeCTE.OrderDate as date) as OrderDate,
            CumulativeCTE.TotalDue, CumulativeCTE.Cumulative
     from Purchasing.Vendor
         join CumulativeCTE on Vendor.BusinessEntityID =
                               CumulativeCTE.VendorID
            where CumulativeCTE.PurchaseOrderID =
           (select top 1 PurchaseOrderID from CumulativeCTE Inside
              where VendorID = Vendor.BusinessEntityID
                order by PurchaseOrderID )
  order by VendorID, OrderDate
```

window) of rows. Second, the PARTITION BY VendorID tells SQL Server that aggregated sum based on the "group" of VendorID. Third, the ROWS UNBOUNCED PRECEDING tells SQL Server to start at the first row (based on the PARTITION and in order by PurchaseOrderID) and sum up to the current row. This is how you tell SQL Server, "for the current row in the result set, sum all the prior rows for that vendor, starting with the first row and ending with the current row." This is powerful stuff!

Listing 3 shows the full-solution counterpart to **Listing 1**. You still need to do some type of SELF JOIN to dive back into this table to grab the first row where the cumulative amount exceeds 10,000. It's very difficult to do all of this in one simple query, but at least the new language statements in SQL 2012 make this simpler than prior versions.

3: Moving Averages in SQL 2012

Programming challenge: As shown in **Figure 2**, based on Orders in AdventureWorks in 2006, write a query to retrieve the weekly orders and four-week moving average of weekly orders for a specific vendor. The "current week" should be included in the four-week range. If a week has no sales (null), you include it (as if the sales were zero).

Fortunately, you can use the same windowing functions and features in SQL Server 2012 that you saw in the last tip. In this instance, for each summarized row (by week), you can average "over" the four prior rows using the syntax in the snippet below. Note that instead of using ROWS UNBOUNDED PRECEDING as you did before, you use ROWS 3 PRECEDING to include the three prior rows based on the order of the Week Ending Date. Because the AVG OVER includes the current row, retrieving the three prior rows gives us the four-week moving average effect. Also note that you use ISNULL to transform any NULL values to zero. Otherwise, aggregation functions like AVG would ignore the NULL week (unless that's the intention).

```
select  DateList.WeekEnding, SumTotalDue,
   AVG( ISNULL(SumTotalDue,0)) OVER
       (ORDER BY DateList.WeekEnding
          ROWS 3 preceding)
   as MovingAvg
 FROM …
```

Listing 4 shows the entire solution, which also includes a simple table-valued function to explode a date range between two dates, and also a scalar function to express a date in terms of a week-ending Saturday date.

Listing 2: Cumulative SUM using UNBOUNDED PRECEDING in SQL 2012

```
select PurchaseOrderID, OrderDate, TotalDue,
    SUM(TotalDue) OVER (PARTITION BY VendorID
                        ORDER BY PurchaseOrderID
          ROWS UNBOUNDED PRECEDING  ) AS CumulativeAmount
 from  Purchasing.PurchaseOrderHeader
     where VendorID = 1494 and year(OrderDate) = 2006
         order by PurchaseOrderID
```

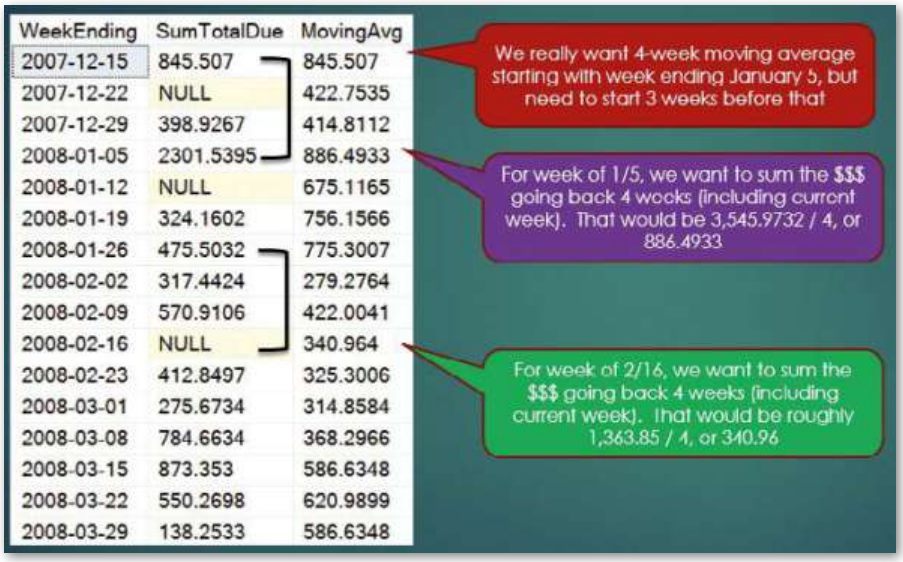


Figure 2: Calculate 4-week moving average

4: Baker's Dozen Spotlight: Calculating Elapsed Time across Rows

Programming challenge: As shown in **Figure 3**, where you have the names of individuals who've moved from phase to phase (for any business process, the specific process doesn't matter) with a specific start and end time, you want to produce a result set that shows the elapsed time from all possible name/starting phases to name/ending phases. For Kevin, you want to know the elapsed time from Phase 1 through Phase 4 (1, 2, 3, 4), and then from Phase 2 to 4 (2, 3, 4), etc. The desired result set is shown in **Figure 4**. Ultimately, you might want to produce average time statistics on how long it takes to move from Phase 2 to Phase 4, etc.

Not too long ago, I had to generate a result set like this for a client. I'll admit freely that I initially made the problem (and the query) too complicated. After some discussions with other developers, I realized all that was needed was a SELF-JOIN into the original table, matching up on the name and where the phase was one less. Once I joined up the outer table on a name/phase with an inner table on the name and "next" phase, I could use the DATEDIFF function to determine the difference between the time for the next phase and the time for the current phase. **Listing 5** shows the entire solution. I've also included an illustration of this solution in **Figure 5:**

5: UNPIVOT versus UNION

Programming challenge: As shown in **Figure 6**, convert the flat, denormalized rows of Monthly Sales by salesmanID to a normalized set of rows.

Name	Phase	StartTime	EndTime
Kevin	Phase 1	2014-03-11 12:57:18.380	2014-03-11 12:59:18.380
Kevin	Phase 2	2014-03-11 12:59:18.380	2014-03-11 13:09:18.380
Kevin	Phase 3	2014-03-11 13:09:18.380	2014-03-11 13:15:18.380
Kevin	Phase 4	2014-03-11 13:15:18.380	2014-03-11 13:21:18.380
Steve	Phase 1	2014-03-11 12:57:49.380	2014-03-11 13:02:18.380
Steve	Phase 2	2014-03-11 13:02:18.380	2014-03-11 13:09:18.380
Steve	Phase 3	2014-03-11 13:09:18.380	2014-03-11 13:16:18.380
Steve	Phase 4	2014-03-11 13:16:18.380	2014-03-11 13:23:18.380

Figure 3: Starting data

Most developers would likely write this solution as a UNION ALL statement, with one SELECT statement for each column. Although this works, developers are likely to include a performance penalty with all the SELECT and UNION ALL statements. UNION ALL won't perform a duplicate check, but an alternate solution might be to use the UNPIVOT feature that Microsoft added in SQL Server 2005. Admittedly, when Microsoft added PIVOT and UNPIVOT in SQL 2005, I first expected that I'd use PIVOT frequently. As it turns out, I've used UNPIVOT more often, especially in the last few years.

UNPIVOT isn't necessarily intuitive and I'll also admit that I need to keep an example close by for when I need it again. But it can give you better performance and a less-costly execution plans, especially when dealing with a high number of rows and columns. **Listing 6** shows a basic example of UNPIVOT to convert the columns to rows. Note that just like with the PIVOT statement, the column list is static—you'd need to use dynamic SQL if you don't know the column names until runtime.

6: The Multiple Aggregation Pattern (and Anti-pattern)

Programming challenge: As shown in **Figure 7**, you have three tables, a Job Master and then two child tables (Job Hours and Job Materials). The relationship between Job Master and the two child tables is one-to-many. You want to produce a result set with one row per job, summarizing the hours and materials.

I've used this example for years (close to a decade) as an example of the dangers of multiple aggregations, and

T-SQL is Dead. Long Live T-SQL!

In the middle of the last decade, I remember hearing people discuss the imminent demise of T-SQL in favor of other tools. Fortunately, it never came true. Good T-SQL skills (and a strong understanding of the internals of the database engine) are still highly valuable. Good chops never go out of style.

name	FromPhase	ToPhase	StartTime	EndTime	Elapsed
Kevin	Phase 1	Phase 1	2014-03-11 12:57:18.380	2014-03-11 12:59:18.380	120
Kevin	Phase 1	Phase 2	2014-03-11 12:57:18.380	2014-03-11 13:09:18.380	720
Kevin	Phase 1	Phase 3	2014-03-11 12:57:18.380	2014-03-11 13:15:18.380	1080
Kevin	Phase 1	Phase 4	2014-03-11 12:57:18.380	2014-03-11 13:21:18.380	1440
Kevin	Phase 2	Phase 2	2014-03-11 12:59:18.380	2014-03-11 13:09:18.380	600
Kevin	Phase 2	Phase 3	2014-03-11 12:59:18.380	2014-03-11 13:15:18.380	960
Kevin	Phase 2	Phase 4	2014-03-11 12:59:18.380	2014-03-11 13:21:18.380	1320
Kevin	Phase 3	Phase 3	2014-03-11 13:09:18.380	2014-03-11 13:15:18.380	360
Kevin	Phase 3	Phase 4	2014-03-11 13:09:18.380	2014-03-11 13:21:18.380	720
Kevin	Phase 4	Phase 4	2014-03-11 13:15:18.380	2014-03-11 13:21:18.380	360
Steve	Phase 1	Phase 1	2014-03-11 12:57:49.380	2014-03-11 13:02:18.380	269
Steve	Phase 1	Phase 2	2014-03-11 12:57:49.380	2014-03-11 13:09:18.380	689
Steve	Phase 1	Phase 3	2014-03-11 12:57:49.380	2014-03-11 13:16:18.380	1109
Steve	Phase 1	Phase 4	2014-03-11 12:57:49.380	2014-03-11 13:23:18.380	1529
Steve	Phase 2	Phase 2	2014-03-11 13:02:18.380	2014-03-11 13:09:18.380	420
Steve	Phase 2	Phase 3	2014-03-11 13:02:18.380	2014-03-11 13:16:18.380	840
Steve	Phase 2	Phase 4	2014-03-11 13:02:18.380	2014-03-11 13:23:18.380	1260
Steve	Phase 3	Phase 3	2014-03-11 13:09:18.380	2014-03-11 13:16:18.380	420
Steve	Phase 3	Phase 4	2014-03-11 13:09:18.380	2014-03-11 13:23:18.380	840
Steve	Phase 4	Phase 4	2014-03-11 13:16:18.380	2014-03-11 13:23:18.380	420

Figure 4: Final result set

Listing 3: Full example of ROWS UNBOUNDED PRECEDING in SQL 2012

```
;with CumulativeCTE AS
 (select VendorID, PurchaseOrderID, OrderDate, TotalDue,
    SUM(TotalDue) OVER (PARTITION BY VendorID ORDER BY
                        PurchaseOrderID
              ROWS UNBOUNDED PRECEDING  ) AS CumulativeAmount
 from  Purchasing.PurchaseOrderHeader
 WHERE OrderDate BETWEEN '1-1-2006' AND '12-31-2006' )

 SELECT Vendor.Name, Outside.*
 FROM Purchasing.Vendor
  JOIN CumulativeCTE Outside
          ON Vendor.BusinessEntityID = Outside.VendorID
     WHERE Outside.PurchaseOrderID = (SELECT TOP 1 PurchaseOrderID
                                      FROM CumulativeCTE Inside
                                       WHERE Inside.VendorID =
                                        Outside.VendorID AND
                                       Inside.CumulativeAmount > 10000
                                      ORDER BY Inside.PurchaseOrderID)
```

Listing 4: Moving average in SQL 2012 with AVG OVER

```
CREATE FUNCTION [dbo].[CreateDateRange]
(@StartDate Date, @EndDate Date)
RETURNS
@WeekList TABLE (WeekEnding Date)
AS
BEGIN

;WITH DateCTE(WeekEnding) AS
      (SELECT @StartDate AS WeekEnding
               UNION ALL
                  SELECT DateAdd(d,7,WeekEnding) AS WeekEnding
                      FROM DateCTE WHERE WeekEnding < @EndDate
               )

  insert into @WeekList  select * from DateCTE
        OPTION (MAXRECURSION 1000)

RETURN
END

GO


CREATE FUNCTION [dbo].[WeekEndingDate]
( @InputDate date  )
RETURNS Date
AS
BEGIN
     DECLARE @ReturnDate DATE
    -- @@DateFirst defaults to 7  (Sunday),
     SET @ReturnDate =
           dateadd(day,
           (@@DateFirst -
              datepart(weekday,@InputDate)),
                                    @InputDate)
     RETURN @ReturnDate
END
GO

DECLARE @StartDate DATE, @EndDate DATE
SET @StartDate = '12-8-2007'
SET @EndDate = '3-29-2008'

;With summarizedCTE as
(select VendorID, dbo.WeekEndingDate(OrderDate) as WeekEndDate,
      SUM(TotalDue) AS SumTotalDue
     FROM  Purchasing.PurchaseOrderHeader POH
       WHERE OrderDate BETWEEN @StartDate  AND @EndDate
       AND VendorID = 1496
       GROUP BY VendorID, dbo.WeekEndingDate(OrderDate))


select  DateList.WeekEnding, SumTotalDue,
   AVG( ISNULL(SumTotalDue,0)) OVER
       (ORDER BY DateList.WeekEnding ROWS 3 preceding) as MovingAvg
 FROM
    dbo.CreateDateRange ( @StartDate, @EndDate ) DateList
 left outer join  SummarizedCTE on DateList.WeekEnding =
                  summarizedCTE.WeekEndDate
                   order by WeekEnding
```

where subqueries are necessary. Although I've tried to not use examples from prior Baker's Dozen articles, in this article, I admit that I've used this before. However, as a consultant, I continue to see developers who (understandably) fall victim to the anti-pattern I'm about to describe. And admittedly, a few weeks ago, I almost caught myself falling into the trap myself!

Listing 7 first shows a standard SELECT with two LEFT OUTER JOINSs into the two child tables, with aggregations on both the Hours Worked and the Purchase Amounts. If you weren't aware of the problem, you might think the first query would be acceptable. Unfortunately, it generates the results shown in **Figure 8**.

You should have had values of 30 hours and $3,000 for Job A, but instead, the results show 60 hours and $12,000. Is this bug in SQL Server? Absolutely not. This is a case of the database engine doing precisely what you

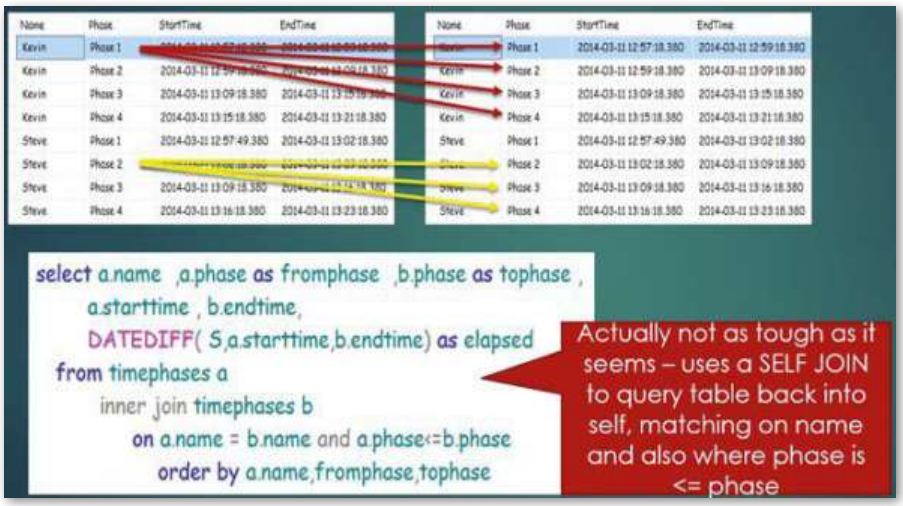


Figure 5: Using the SELF JOIN

SalesManID	Jan	Feb	Mar
Salesman A	100.0000	200.0000	300.0000
Salesman B	400.0000	500.0000	600.0000
Salesman C	700.0000	800.0000	NULL
Salesman D	900.0000	NULL	NULL

Product this result

SalesManID	MonthName	SalesForMonth
Salesman A	Jan	100.0000
Salesman A	Feb	200.0000
Salesman A	Mar	300.0000
Salesman B	Jan	400.0000
Salesman B	Feb	500.0000
Salesman B	Mar	600.0000
Salesman C	Jan	700.0000
Salesman C	Feb	800.0000
Salesman D	Jan	900.0000

Figure 6: Using the SELF JOIN

JobNumber	JobDescription
1	Job A
2	Job B
3	Job C
4	Job D

JobNumber	EmployeeID	DateWorked	HoursWorked
1	1	2008-01-01 ...	8.00
1	1	2008-01-02 ...	8.00
1	2	2008-01-01 ...	7.00
1	2	2008-01-02 ...	7.00
2	3	2008-01-01 ...	7.00
2	3	2008-01-02 ...	7.00
2	4	2008-01-01 ...	6.00
2	4	2008-01-02 ...	6.00

JobNumber	Notes	purchaseamount	PurchaseDate
1	bought tractor	1000.00	2008-01-01 ...
1	bought cement	2000.00	2008-01-01 ...
3	bought tractor	10000.00	2008-01-01 ...
3	bought cement	20000.00	2008-01-01 ...

Read the 3 tables on the left (JobMaster, JobHours, and JobMaterials). Produce a result set with 1 row per job that summarizes hours and purchase amounts. Include all jobs, even if child data doesn't exist.

This appears simple, but is a "trap" if you try to do it all in one SELECT with multiple aggregations into multiple 1-many tables

JobDescription	TotHours	TotPurchase
Job A	30.00	3000.00
Job B	26.00	NULL
Job C	NULL	30000.00
Job D	NULL	NULL

Figure 7: Querying multiple one-to-many relationships at once

JobDescription	Sum of Hours	Sum of Materials
Job A	60.00	12000.00
Job B	26.00	NULL
Job C	NULL	30000.00
Job D	NULL	NULL

Job A has hours double-counted and materials quadruple-counted (there were 4 Job Hours rows and 2 Job Materials rows)

Figure 8: Incorrect results

TOLD IT to do, as opposed to what you WANTED IT to do. So why are the numbers so inflated for Job A? Before I get to that, here are some hints:

- You should have had 30 hours and we got 60 hours. That's a factor of two.
- You should have had $3,000 for materials and you got $12,000. That's a factor of four.
- There were two rows in Job Hours for Job A.
- There were four rows in Job Materials for Job A.

Basically, the factor by which you were off in the one child table happens to be the number of rows in the other child table. Is this a coincidence? No, it's not! Here's the issue: although unintentional, you've produced an internal Cartesian product (cross join) between the four instances of Job A in hours with the two instances of Job A in job materials. Basically, Job A was "exploded" into eight rows in the internal result set!

This is a perfect example where you can't do all of this in one SELECT statement and where subqueries are necessary. The second query in **Listing 7** uses two common table expressions (derived table subqueries) to pre-aggregate the two child tables. You can then join the master table to the two common table expressions in a one-to-one fashion and generate the correct results.

> Rarely does someone discover and realize an anti-pattern in one shot. The realization starts with identifying some undesirable result—anything from wrong numbers on a report to an inelegant process.

I came across the "multiple aggregations" anti-pattern a little over 15 years ago. I wrote a query similar to the first query in **Listing 7**. Fortunately, I noticed the problem with the results before I implemented the code to production. But I've known developers who weren't so fortunate. Some didn't notice and verify the results, and got bit in production. Rarely does someone discover and realize an anti-pattern in one shot. The realization starts with identifying some undesirable result—anything from wrong numbers on a report to an inelegant process. My first inking on the "multiple aggregations" anti-pattern was noticing inflated numbers in one of my test cases. It took me a while to fully understand the problem, but at least I was fortunate that I caught the problem before the users did!

7: Differences between IN and EXISTS

Programming challenge: Describe the differences between IN and EXISTS.

IN and EXISTS might initially look similar, but in reality they're very far apart. **Listing 8** shows an example for both. Here are the fundamental differences.

- With the IN feature, you use a derived table subquery that could execute independently. You specify a single column on the outside and match to the single column returned on the inside. It's good for general testing but many recommend against it (or caution against it) in production.
- With EXISTS, you use a correlated subquery. The subquery cannot execute independently, as it refers to (correlates with) values from the outer parent query. Unlike the IN statement, you don't specify a specific column on the outside to match with the values returned from the SELECT on the inside. Instead, you match up columns as a WHERE clause on the inside. This is arguably a bit more complicated, but ultimately provides more flexibility

Listing 5: Elapsed time across rows (Baker's Dozen Spotlight)

```
CREATE TABLE TimePhases   ( Name varchar(50), Phase varchar(10),
          StartTime datetime, EndTime datetime)

INSERT INTO TimePhases values ('Kevin', 'Phase 1',
 '2014-03-11 12:57:18.380', '2014-03-11 12:59:18.380')

INSERT INTO TimePhases values ('Kevin', 'Phase 2',
  '2014-03-11 12:59:18.380', '2014-03-11 13:09:18.380')

INSERT INTO TimePhases values ('Kevin', 'Phase 3',
  '2014-03-11 13:09:18.380', '2014-03-11 13:15:18.380')

INSERT INTO TimePhases values ('Kevin', 'Phase 4',
   '2014-03-11 13:15:18.380', '2014-03-11 13:21:18.380')


INSERT INTO TimePhases values ('Steve', 'Phase 1',
    '2014-03-11 12:57:49.380', '2014-03-11 13:02:18.380')

INSERT INTO TimePhases values ('Steve', 'Phase 2',
     '2014-03-11 13:02:18.380', '2014-03-11 13:09:18.380')

INSERT INTO TimePhases values ('Steve', 'Phase 3',
     '2014-03-11 13:09:18.380', '2014-03-11 13:16:18.380')

INSERT INTO TimePhases values ('Steve', 'Phase 4',
     '2014-03-11 13:16:18.380', '2014-03-11 13:23:18.380')



 select a.name  ,a.phase as fromphase  ,b.phase as tophase ,
        a.starttime , b.endtime,
        DATEDIFF( S,a.starttime,b.endtime) as elapsed
          from timephases a
            inner join timephases b
                  on a.name = b.name and a.phase<=b.phase
                     order by a.name,fromphase,tophase
```

Listing 8 shows the difference between IN and EXISTS. **Listing 9** also shows another issue with IN—or more specifically, with NOT IN. A developer might want to know how many keys are "not in" the results from a derived table subquery. The problem is that if the derived table subquery contains any NULL values for the column being used in the match, the outer result will always be empty! As a result, you need to make sure that you include a WHERE <column> IS NOT NULL in the derived table subquery (or just use the recommended EXISTS instead).

> You'd be amazed at how much you can take aspects of a feature for granted until you really start breaking down definitions.

Recently I asked a group of SQL developers to verbally describe the difference between IN and EXISTS. I knew they'd written code for those features before, but was curious how they'd respond. It might seem like an academic exercise, but try to take some feature you frequently use and define it with a limited number of references to specific code. You'd be amazed at how much you can take aspects of a feature for granted until you really start breaking down definitions. (I'm the King of taking things for granted if I don't stop to think about them!)

8: A Use for RANK Functions

Programming challenge: Using AdventureWorks purchase orders in 2006 summarized by Vendor and Ship Method, show each Ship Method and the top two Vendor IDs (in terms of total order dollars).

Although some developers might use the **SELECT TOP 2 VendorID order by DollarAmount** approach, you'd need to use it for each Ship Method. You can do that, but another approach is to use the RANK functions that Microsoft introduced in SQL Server 2005. **Listing 10** contains a query that generates a rank number (within each Ship Method, ordered by the sum of order dollars). Here's the core code that uses the RANK function and PARTITION BY/ORDER BY statements:

Listing 6: UNPIVOT as an alternative to UNION ALL

```
create table dbo.SalesTemp
  (SalesManID varchar(10), Jan decimal(14,4), Feb decimal(14,4),
    Mar decimal(14,4))
GO

insert into SalesTemp values ('Salesman A', 100, 200, 300),
                             ('Salesman B', 400, 500, 600),
                             ('Salesman C', 700, 800, null),
                             ('Salesman D', 900, null, null)

SELECT SalesManID, MonthName, SalesForMonth
  FROM dbo.SalesTemp  AA
   UNPIVOT (SalesForMonth FOR MonthName IN (Jan, Feb, Mar)
           ) TempData
```

```
SELECT ShipMethodID, VendorID,
       SUM(TotalDue) as TotalDollars,
       RANK() OVER  (PARTITION BY ShipMethodID
        ORDER BY SUM(TotalDue) DESC) as VendorRank
 FROM Purchasing.PurchaseOrderHeader
```

Of course, that code snippet returns all ShipMethodID/VendorID combinations with orders. You want the first two vendors for each Ship Method. Can you use the VendorRank alias in the WHERE clause? No,-because you can only use table column names in the WHERE clause. Can you be a bit verbose and repeat the RANK function in the WHERE clause, much in the same way that you might do with a scalar function? No, because the RANK functions read across a set (window) of rows, and the WHERE clause operates row-by-row. So you'd need to wrap that SELECT statement (in the snippet above) as a derived table/common table expression and then refer to the materialized VendorRank column on the outside.

9: Cursors versus Table Types for Performance

Listing 11 and **Listing 12** represent a challenge that developers face when attempting to execute a stored

Listing 7: Danger of multiple aggregations

```
CREATE TABLE JobMaster (JobNumber int, JobDescription varchar(50))
CREATE TABLE JobHours  (JobNumber int, EmployeeID int,
          DateWorked DateTime,   HoursWorked decimal(10,2))
CREATE TABLE JobMaterials  (JobNumber int, Notes varchar(50),
        purchaseamount decimal(14,2),   PurchaseDate DateTime)

insert into JobMaster values (1, 'Job A'),   (2, 'Job B')  ,
                             (3, 'Job C'), (4, 'Job D')


INSERT INTO JobHours values (1, 1, '01-01-08',8)
INSERT INTO JobHours values (1, 1, '01-02-08',8)
INSERT INTO JobHours values (1, 2, '01-01-08',7)
INSERT INTO JobHours values (1, 2, '01-02-08',7)

INSERT INTO JobHours values (2, 3, '01-01-08',7)
INSERT INTO JobHours values (2, 3, '01-02-08',7)
INSERT INTO JobHours values (2, 4, '01-01-08',6)
INSERT INTO JobHours values (2, 4, '01-02-08',6)


insert into JobMaterials values (1, 'got tractor',1000,'1-1-08')
insert into JobMaterials values (1, 'got cement',2000,'1-1-08')
insert into JobMaterials values (3, 'got tractor',10000,'1-1-08')
insert into JobMaterials values (3, 'got cement',20000,'1-1-08')

-- this produces incorrect results for Job A
select jobmaster.JobDescription, SUM(HoursWorked) as SumHours,
                    SUM(purchaseamount) as SumMaterials
      from JobMaster
           left outer join JobHours
                on JobMaster.JobNumber = JobHours.JobNumber
           left outer join JobMaterials
                on JobMaster.JobNumber = JobMaterials.JobNumber
                     Group by JobMaster.JobDescription
                          order by JobDescription


-- By aggregating as separate definitions, we get the right results

; with HoursCTE AS
      (SELECT JobNumber, SUM(HoursWorked) as TotHours
               from JobHours group by JobNumber) ,
      MaterialsCTE AS
               (SELECT JOBNUMBER, SUM(PurchaseAmount) as TotPurchase
                    from JobMaterials GROUP BY JobNumber)

SELECT JobMaster.JobNumber, JobMaster.JobDescription,
      TotHours, TotPurchase
FROM JobMaster
    LEFT join HoursCTE
        ON JobMaster.JobNumber = HoursCTE.JobNumber
    LEFT JOIN MaterialsCTE
         ON JobMaster.JobNumber = MaterialsCTE.JobNumber
```

Listing 8: IN vs EXISTS

```
-- using IN, must match up a single column.  Inside query
-- can run on its own
SELECT Vendor.Name
     from Purchasing.Vendor
         WHERE ( BusinessEntityID) in
           (SELECT VendorID   from Purchasing.PurchaseOrderHeader
                     WHERE YEAR(OrderDate) = 2006 AND
                       ShipMethodID = 1)


-- Correlated subquery
  SELECT VendorOutside.Name
     from Purchasing.Vendor   VendorOutside
        WHERE  EXISTS (  SELECT * from
            Purchasing.PurchaseOrderHeader
             WHERE YEAR(OrderDate) = 2006 AND   ShipMethodID = 1
                AND VendorID = VendorOutside.BusinessEntityID  )
```

procedure for a set of rows. I'll whittle it down to a very simple example, but one where the pattern applies to many situations. The two listings contain code for two different approaches to solve the problem. **Listing 11** uses SQL Server Cursors, an older approach that can sometimes yield serious performance issues. **Listing 12** uses Table Types, a feature that Microsoft added in SQL Server 2008. I'll cover the approaches in reverse order. First, I'll talk about Table Types as I break down the code in **Listing 12**.

Suppose you have a stored procedure that receives a single vendor ID and updates the freight for all orders with that vendor ID. Now, suppose you need to run this procedure for a set of vendor IDs. Today you might run the procedure for three vendors, tomorrow for five vendors, the next day for 100 vendors. Each time, you want to pass in the vendor IDs. If you've worked with SQL Server, you can probably guess where I'm going with this. The big question is: How do you pass a variable number of Vendor IDs? Or, stated more generally, how do you pass an array, or a table of keys, to a procedure? Something along the lines of:

```
exec dbo.UpdateVendorOrders @SomeListOfVendors
```

Prior to SQL Server 2008, developers needed to pass XML strings or comma-separated lists of keys to a procedure, and the procedure converted the XML/CSV strings into table variables for eventual joins with other data. Although this worked, developers wanted a more direct path. In SQL Server 2008, Microsoft implemented the table type. This FINALLY allowed developers to pass an actual table of rows to a stored procedure.

Passing a table of rows to a stored procedure requires a few steps. You can't just pass any old table to a procedure. It has to be a pre-defined type (a template). Let's suppose that you always want to pass a set of integer keys to different procedures. One day it might be a list of vendor keys. The next day it might be a list of customer keys. You can create a generic table type of keys, one that can be instantiated for customer keys, vendor keys, etc.

```
CREATE TYPE IntKeysTT AS TABLE
 ( [IntKey] [int] NOT NULL )
GO
```

I've created a Table Type called **IntKeysTT**. It's defined to have one column: an IntKey. Now, suppose I want to load it with Vendors who have a Credit Rating of 1 and then take that list of Vendor keys and pass it to a procedure:

Listing 9: Danger of NOT IN with NULL values

```
DECLARE @ColorMaster TABLE (ColorID int identity,
                    Color  varchar(100))
DECLARE @SalesTable TABLE  (SalesID int identity,
                   SalesAmount money, ColorID int)

insert into @ColorMaster (Color) values ('Red'), ('Blue'),
                            ('Yellow')
insert into @SalesTable (SalesAmount, ColorID) values ( 100, 1),
                    (200, 2), (300, null)

select * from @ColorMaster Colors where ColorID in (select ColorID
  from @SalesTable )  -- 2 rows....Red and Blue

select * from @ColorMaster Colors where ColorID not in (select
   ColorID from @SalesTable )  --  0 rows - WHAT???


select * from @ColorMaster Colors where ColorID not in
     (select ColorID from @SalesTable  WHERE ColorID is not null)
  --  This works, must filter out the NULLs on the inside
  -- or SQL Server generates zero rows in outer result set
```

```
DECLARE @VendorList IntKeysTT

INSERT INTO @VendorList
   SELECT BusinessEntityID
       from Purchasing.Vendor
         WHERE CreditRating = 1
```

Now I have a table type variable, and not just any table variable, but a table type variable (that I populated the same way I would populate a normal table variable). It's in server memory (unless it needs to spill to tempDB) and is therefore private to the connection/process.

> Now that there's a Table Type feature in SQL Server 2008, you can achieve the objective with a feature that's more closely modeled to the way developers think.

Can you pass it to the stored procedure now? Not yet. You need to modify the procedure to receive a table type. The full code is in **Listing 12**, but here's the part of the stored procedure that you need to modify:

```
create procedure dbo.UpdateVendorOrdersFromTT
  @IntKeysTT IntKeysTT READONLY
    As
      Begin
```

Notice how the procedure receives the parameter IntKeysTT table type as a Table Type (again, not just a regular table, but a Table Type). It also receives the parameter as a READONLY parameter. You CANNOT modify the contents of this table type inside the procedure. Usually you won't want to; you simply want to read from it. Now you can reference the table type as a parameter and then use it in the JOIN statement, just as you would any other table variable.

There you have it. It's a bit of work to set up the table type, but in my view, definitely worth it.

Additionally, if you pass values from .NET, you're in luck. You can pass an ADO.NET data table (with the same tablename property as the name of the Table Type) to the procedure. For .NET developers who've had to pass CSV lists, XML strings, and so forth to a procedure in the past, this is a huge benefit!

Listing 10: RANKing and PARTITIONING

```
SELECT * FROM
(SELECT ShipMethodID, VendorID, SUM(TotalDue) as TotalDollars,
RANK() OVER  (PARTITION BY ShipMethodID
              ORDER BY SUM(TotalDue) DESC) as VendorRank
 FROM Purchasing.PurchaseOrderHeader
    WHERE OrderDate BETWEEN '1-1-2006' AND '12-31-2006'
 GROUP BY  ShipMethodID, VendorID ) Temp
  -- can't filter on RANK, must use subquery and then refer
   -- to the ALIAS column
 WHERE VendorRank <= 2
```

Now I'd like to talk about **Listing 11** and an approach people have used over the years: SQL Server Cursors. At the risk of sounding dogmatic, I strongly advise against Cursors unless there's just no other way. Cursors are expensive operations in the server. For instance, someone might use a cursor approach and implement the solution in the way I'll describe in a moment.

The best thing I'll say about SQL Server Cursors is that it "works". And yes, getting something to work at all is a milestone. But getting something to work and getting something to work well are two different things. Even if this process only takes 5-10 seconds to run, in those 5-10 seconds, the cursor uses SQL Server resources quite heavily, which isn't a good idea in a large production environment. Additionally, the greater the number of rows in the cursor to fetch and the greater the number of executions of the procedure, the slower it will be.

When I ran both processes (the cursor approach in **Listing 11** and then the table type approach in **Listing 12**) against a small sampling of vendors (five vendors), the processing times where 260ms and 60ms, respectively. The table type approach was roughly four times faster. Then when I ran the two scenarios against a much larger number of vendors (84 vendors), the difference was staggering: 6701ms versus 207ms, respectively. The table type approach was roughly 32 times faster.

Again, the CURSOR approach is definitely the least attractive approach. Even in SQL Server 2005, it would have been better to create a CSV list or an XML string (if the number of keys can be stored in a scalar variable). But now that there's a Table Type feature in SQL Server 2008,

The New Way versus the Old Way

Microsoft made plenty of enhancements to T-SQL in SQL Server 2012, particularly the windowing and statistical functions. Even if you have an older code base to solve these problems using pre-SQL Server 2012 features, at least be aware of the new features because you'll see other developers use them.

Listing 11: Cursors versus Table Types (First, Cursors – BAD!!!)

```
create procedure dbo.UpdateVendorOrders
@VendorID int
As
Begin
update Purchasing.PurchaseOrderHeader set Freight = Freight + 1
where VendorID = @VendorID
end
go



create procedure dbo.RunUpdateVendorsUsingCursor
as
begin
set nocount on

DECLARE @VendorID int
DECLARE db_cursor CURSOR FAST_FORWARD FOR
 SELECT BusinessEntityID from Purchasing.Vendor
where CreditRating = 1

OPEN db_cursor
FETCH NEXT FROM db_cursor INTO @VendorID
WHILE @@FETCH_STATUS = 0
BEGIN
   EXEC dbo.UpdateVendorOrders @VendorID
   FETCH NEXT FROM db_cursor INTO @VendorID

END

CLOSE db_cursor

DEALLOCATE db_cursor
end
go


exec  dbo.RunUpdateVendorsUsingCursor    -- takes 48-52 seconds
```

Listing 12: Cursors versus Table Types: (Second, Table Types, GOOD!!!)

```
CREATE TYPE IntKeysTT AS TABLE
 ( [IntKey] [int] NOT NULL )
GO



create procedure dbo.UpdateVendorOrdersFromTT
@IntKeysTT IntKeysTT READONLY
As
Begin

 update Purchasing.PurchaseOrderHeader set Freight = Freight + 1
    FROM Purchasing.PurchaseOrderHeader
            JOIN @IntKeysTT TempVendorList
     ON PurchaseOrderHeader.VendorID = TempVendorList.IntKey
End
go



create procedure dbo.RunUpdateVendorsUsingTTs
as
begin
DECLARE @VendorList IntKeysTT
INSERT INTO @VendorList
 SELECT BusinessEntityID from Purchasing.Vendor
WHERE CreditRating = 1

 exec dbo.UpdateVendorOrdersFromTT  @VendorList
 end
go

exec  dbo.RunUpdateVendorsUsingTTs   -- runs in roughly 1 second
```

Listing 13: Simple (but not so simple) Aggregation

```
select VendorID,
    DATEDIFF(D,MIN(OrderDate), MAX(OrderDate) )
          /  ( COUNT(*)-1)  -- subract 1 for # of intervals
             AS AvgDaysInBetween
from Purchasing.PurchaseOrderHeader
GROUP BY VendorID
HAVING COUNT(*) > 1  -- only for vendors with more than 1 order
ORDER BY VendorID
```

Vendorid	OrderDate
1636	2007-12-05
1636	2008-02-28
1636	2008-06-14
1636	2008-07-25

Figure 9: Four orders for Vendor ID 1636

you can achieve the objective with a feature that's more closely modeled to the way developers think. Specifically: "How do I pass a table to a procedure?" Now you have an answer!

10: Calculating Aggregations across Rows

Programming challenge: Using AdventureWorks, by Vendor, list the average amount of time between orders. Although this could be done by taking the specific time between each individual order, I'll keep this simple. For each vendor, get the earliest order date and the most recent order date, calculate the difference in days, and then divide by the number of orders (minus 1, since a vendor with two orders would only have one segment of time between them).

I'll start by profiling the orders for a small vendor, Vendor ID 1636 (**Figure 9**). Because the rules state that you need to take the oldest and newest dates and divide by the number of orders, you can use MIN and MAX, subtract one from the other to determine the total number of days between, and then divide by the count of rows in order to come up with a basic average.

The total number of days elapsed for this vendor (between the first and last orders) is 233 days. If you divide that by four, you get 58.25 days. You know that number can't be right, as the difference between two of the date ranges far exceeds 58.25 days. You need to divide by three (or the COUNT(*) minus 1), as you actually have three periods in between the four orders (from 1 to 2, from 2 to 3, and from 3 to 4). The final result in **Listing 13** needs to divide the total elapsed days by the count, less 1. That gives you the correct results (as seen in **Figure 10**).

Also note that in **Listing 13**, you need to use a HAVING clause to only include those vendors with a total order

count greater than 1. If a vendor has only one order, there's no elapsed time and therefore nothing to calculate.

Finally, suppose that you wanted to change this exercise and produce a result set that showed the current order and "last order date" on the same line. You can use the LAG function to "hop back" one row, based on the order sequence of the order date, and within the VendorID group/partition.

VendorID	AvgDaysInBetween
1636	77
1638	18
1644	18
1646	18

Figure 10: Final results

Vendorid	OrderDate	LastOrderD...	DaysInBetween
1636	2007-12-05	NULL	NULL
1636	2008-02-28	2007-12-05	85
1636	2008-06-14	2008-02-28	107
1636	2008-07-25	2008-06-14	41

Figure 11: Row by row of order date and the Previous Order Date

Listing 14: Parameter Sniffing, the basics

```
select PurchaseOrderID, PurchaseOrderDetailID, OrderQty,
       UnitPrice, LineTotal
  from Purchasing.PurchaseOrderDetail
    where ProductID = 394  -- 19 rows   - Index seek + key lookup
    -- where ProductID = 492  --  63 rows  - Index scan
    -- where ProductID = 319  -- 130 rows - Index scan


create procedure dbo.GetOrdersForSingleProduct
 @ProductID int
 as
   begin
   select PurchaseOrderID, PurchaseOrderDetailID,
          OrderQty, UnitPrice, LineTotal
        from Purchasing.PurchaseOrderDetail
            where ProductID = @ProductID
       -- option (recompile)

end
go


exec dbo.GetOrdersForSingleProduct @ProductID = 394 with recompile
exec dbo.GetOrdersForSingleProduct @ProductID = 492 with recompile
exec dbo.GetOrdersForSingleProduct @ProductID = 319 with recompile
```

Listing 15: Stored Procedures that try to run for one value or ALL

```
create procedure GetOrdersForVendor
@VendorID int
as
 select PurchaseOrderID, OrderDate, ShipMethodID, TotalDue
    FROM  Purchasing.PurchaseOrderHeader
where @VendorID is null or  VendorID = @VendorID



--We get an index scan for both???? even for one vendor???
exec GetOrdersForVendor  @VendorID = 1492
exec GetOrdersForVendor @VendorID = null


alter procedure GetOrdersForVendor
@VendorID int
as
select PurchaseOrderID, OrderDate, ShipMethodID, TotalDue FROM
      TempPurchaseOrderHeader
WHERE  VendorID = @VendorID or @VendorID is null
option (recompile)  -- statement level recompile
go
```

```
select *,
  datediff(d,LastOrderDate, OrderDate)
   as DaysInBetween
FROM
 (select Vendorid,
    cast(orderdate as date) as OrderDate,
       lag(cast( orderdate as date),1,null) over
          (Partition by VendorID
           Order by OrderDate) as LastOrderDate
 from Purchasing.PurchaseOrderHeader  ) Temp
 order by orderdate
```

That produces a result set like the one in **Figure 11**.

11: Parameter Sniffing and Stored Procedure Execution Plans

Listing 14 shows the basics of parameter sniffing. Suppose that you have a table with an index on Product ID. The three stand-alone queries at the beginning of **Listing 14** yield at least two different execution plans. For one value, the query generates an execution plan with an **Index Seek** execution operator plus a **Key Lookup** execution operator (to retrieve the non-key columns) and the other values generate an execution plan with an **Index Scan**. In the latter case, it's because SQL Server examined the distribution of values of the product ID within the clustered (physical order) index and determined that an **index Scan** would be more efficient than an **Index Seek** and a subsequent **Key Lookup**.

> You can use the LAG function to "hop back" one row, based on the order sequence of the order date, and within the VendorID group/partition.

This is standard (and expected) SQL Server behavior. However, if you take the query and define it as a parameterized stored procedure, the behavior changes. If you try to execute the stored procedure for each of the three key product ID values, you'll get an **Index Scan** every time, which otherwise was NOT the optimal plan when you ran the queries independently (i.e., not as stored procedures).

This is the fundamental issue of parameter sniffing. SQL Server "sniffs out" an execution plan upon the first execution and uses that plan. It doesn't generate a plan dynamically every time. EXCEPT if you use a statement level WITH (RECOMPILE) in the query (or by using a WITH RECOMPILE when you execute the procedure). This generates a new execution plan dynamically every time (and usually gives you the desired plan). However, it means that SQL Server needs to rebuild the execution plan every time, which might be highly undesirable if the database must rebuild the plan many times during the day (or hour, or even minute). The developers and DBAs will need to determine the proverbial "pain point" level.

12: Stored Procedures that Try to Query for One Parameter Value or ALL

Listing 15 demonstrates a problem that isn't unlike the issue in the previous tip. Suppose you have a stored procedure that returns orders for a vendor but sometimes you want to pass a NULL value and retrieve all the orders?

You might use something like this in the query:

```
where @VendorID is null or  VendorID = @VendorID
```

I've known developers who thought that SQL Server would be smart enough to execute an **Index Seek** if they passed in a value into the stored procedure, and only used an **Index Scan** if they passed in a NULL. I've also known a few developers who believed that if they changed the logic in the WHERE (re-arranged the operators), SQL Server would generate the best execution plan, given the condition and the least amount of runtime evaluation that SQL Server could do. Unfortunately, both of those beliefs are incorrect. SQL Server generates one execution plan to cover both conditions and does so conservatively. As was the case in the prior tip, you'd need to use a statement-level RECOMPILE.

13: The Difference between WHERE and HAVING

I'll close with a fairly easy one. You use WHERE when operating on single rows from the tables/views you're querying, and you use HAVING in conjunction with aggregations. **Listing 16** shows a query that retrieves all vendor orders in 2006 and only shows the vendor if the sum total of orders in 2006 exceeds $200,000 in sales.

You use the WHERE clause to filter out the individual orders that weren't in 2006, and the HAVING statement to filter on just those vendors who aggregate total sales (given the parameters of the query) exceeded $200.000. Note that you can't use the AnnualTotal alias in the HAVING because you need to repeat the expression.

You use HAVING in conjunction with aggregations, which leads to a small and subtle point: Might the following query work?

```
SELECT  SUM(TotalDue) as AnnualTotal
FROM Purchasing.PurchaseOrderHeader
WHERE OrderDate BETWEEN '1-1-2006' AND '1-2-2006'
HAVING SUM(TotalDue) > 4000
```

Because the query doesn't have a GROUP BY, some might be tempted to say that the query won't work. Actually, it does. In this instance, the query either returns one row (the grand total of sales for the first two days of 2006), or an empty result set (if the grand total in the first two days didn't exceed 4,000). Those who say that HAVING works when you have a GROUP BY aren't 100% correct. It's true that *most* aggregation queries use a GROUP BY, but it's not *always* true. A HAVING still works as long as there's an aggregation.

Final Thoughts:

I hope you're able to gain some benefit from the T-SQL tips here. As I said at the beginning, experienced developers are probably aware of much of the content I've written here. But I hope that new- and intermediate-level developers will pick up some worthwhile knowledge.

Listing 16: The difference between WHERE and HAVING

```
SELECT VendorID, SUM(TotalDue) as AnnualTotal
FROM Purchasing.PurchaseOrderHeader
-- WHERE works on individual rows from the table(s)
WHERE OrderDate BETWEEN '1-1-2006' AND '12-31-2006'
GROUP BY VendorID
-- HAVING works on the aggregated amounts
HAVING SUM(TotalDue) > 200000
```

Whether you're a .NET developer, an ETL developer, an SSRS developer, or a data warehouse specialist, everyone has to write T-SQL code at some point. This is truly one of the "ties that bind us."

Some Anniversaries

My world changed forever in May 1987. I started my career 28 years ago, at the same time I watched my younger brother graduate from basic training in the U.S. Marine Corps. The events of that month shaped our lives forever. EDS paid me about a thousand dollars to write a database application. At night, I read every word in trade publications like Dr. Dobbs Journal, PC Tech Journal, and I witnessed the power of the Compaq DeskPro 386 and software tools like dBase and Microsoft/Turbo C. As someone whose initial field of study was liberal arts, I knew the power of words and ideas and the value of practical education. So much of what I've done since then, and especially over the last decade as a SQL Server MVP, as an instructor and a technical mentor, is rooted in those early days when I saw the explosion of the PC industry and how much I wanted to be a part of it. And like the good marine my brother has always been, I've always been a fanatic about it.

Additionally, in May 2005, I started putting together ideas for what became my first article in CODE Magazine. The great writer and DDJ columnist Al Stevens had always been a huge inspiration for me. Fast forward 11 years and nearly 50 Baker's Dozen articles later, I've been able to carve out the path I envisioned, thanks to CODE Magazine and (in particular) Rod Paddock for supporting the method to my madness.

I thought it would be the coolest thing in the universe to be part of both the software and the magazine industries all those years ago. I hadn't yet experienced the waywardness of business and the cynicism it sometimes breeds. But that's what good memory is for, to remind myself of all the things that helped shape me. I try to remember where I came from, and always keep my eyes on where I want to go.

Kevin S. Goff

CODE

Set-based Approaches

Sometimes it can be very tempting to build a row-by-row solution to a database challenge. The Database cursor is like the million-calorie food item you know you shouldn't eat. Stick to the steady diet of set-based approaches. You might have to spend time on the database treadmill to shape up a good solution, but in the end it's worth it.

XAML Anti-Patterns: Layout SNAFUs

One of the great features of all XAML dialects (and their underlying technologies) is the awesome power that comes with the provided layout engine. Unlike in many of the other UI development technologies, all XAML variations (WPF, Universal Windows Apps, etc.) provide a rich and flexible way of laying out elements on the screen in ways that range from simple,

Markus Egger
markus@eps-software.com

Markus is the founder and publisher of CODE Magazine and is EPS President and Chief Software Architect. He is also a Microsoft RD (Regional Director) and the one of the longest (if not THE longest) running Microsoft MVP (Most Valuable Professional). Markus is also a renowned speaker and author.

Markus spends most of his time writing production code. Markus' client list contains some of the world's largest companies, including many on the Fortune 500. He's worked as a contractor for Microsoft (including the Visual Studio team) and presented at local user groups and major events, such as MS TechEd. Markus has been published extensively including MSDN Magazine, Visual Studio Magazine, his own CODE Magazine, and much more. Markus is a supporter of communities in North America, Europe, and beyond.

Markus focuses on development in .NET (Windows, Web, Windows Phone, and WinRT) as well as Android and iOS. He is passionate about overall application architecture, SOA, user interfaces, general development productivity, and building maintainable and reusable systems.

hand-coded UIs to completely automated and customized approaches. Where other systems provide few or no choices to lay out elements on the screen (such as WinForm's fixed position and size approach), and others offer just a handful of options (HTML's flow layout "with variations" comes to mind), XAML technologies provide a very rich layout system. You can position elements in absolute terms inside of Canvas or Grid containers. You can opt for flow or stack layouts and even lay out rich text documents. And it's even possible to create custom layout approaches that lay out screens for business applications (for instance, consider my article "Super Productivity: Using WPF and Silverlight's Automatic Layout Features in Business Applications" at http://www.codemag.com/Article/1011071, which is just as applicable in today's XAML dialects as it was a few years ago when I wrote the article).

When used correctly, the layout system is the very feature that makes XAML more productive than other UI technology and using it incorrectly qualifies as an anti-pattern. That's what takes me to the current installment of this article series.

> When used correctly, the layout system is the very feature that makes XAML more productive than other UI technologies.

Layout Fundamentals

The basic idea behind "layout" (the act of positioning elements on the screen) in XAML follows a pretty simple pattern: UI elements are placed inside of other UI elements (containers). The container is the element that decides where its child elements are positioned. It does this based on its own rules, but also by taking the properties and attributes of the child element into consideration. This process is repeated ad-nauseam in a hierarchical fashion, from the largest of elements down to the smallest controls.

Let's look at a few simple examples. The root of many XAML apps is an object such as a Window or a Page. Let's pick WPF's Window UI element as an example. Inside of a Window, you can place exactly one child element. (Containers come in two variations: those that contain a single element and those that contain multiple elements). For instance, you could put a Button inside the Window. The code for such a set up would look like this (namespace shortened to keep the example readable):

```
<Window xmlns="http://..." >
  <Button/>
</Window>
```

As you can see, I haven't defined a placement or a size for the button. Instead, the button's position is automatically set by the window object. Because the window object doesn't really know what to do with the button, it makes the only choice that makes sense for any object that's limited to containing a single child: It gives the button the entire space available within the window. In other words, the button's size fills 100% of the available space in the window, no matter how big the window is. When the user resizes the window, the button is resized as well.

You could provide a bit more information to the layout engine. For instance, you could set the Margin property of the button. If you set the margin to 100 pixels on all four sides, the button takes up the available space, minus the defined margin, and it will continue to do so even after the user resizes the window.

> I see few correct uses of the layout system in the wild.

The Margin property is only one example for a property that impacts layout. The point is that it's up to the container (the Window object in this case) to define where its children go. The child elements may have properties set that impact the placement, but it's still up to the container to decide where child elements go.

Of course, most scenarios call for more complex user interfaces than windows that are fully occupied by a single giant button. Instead, the generally accepted approach is to put a different element into the window that, in turn, can contain an unlimited amount of child elements and perform a certain kind of layout. This is Microsoft's way of nudging us toward the idea that the first decision that's made in UI development is which root layout approach to choose. A very common approach is to put a (confusingly named) Grid element into the window and then add the child elements inside that object:

```
<Window xmlns="http://..." >
  <Grid>
    <Button Content="Hello" Margin="10,10,0,0"
            VerticalAlignment="Top"
            HorizontalAlignment="Left"/>
    <Button Content="World" Margin="10,40,0,0"
            VerticalAlignment="Top"
            HorizontalAlignment="Left" />
```

```
  </Grid>
</Window>
```

In this set up, the window hosts the grid and gives it 100% of the available space (just like the button got 100% of the available space). The grid, in turn, decides where the two buttons go that are inside of the grid. The grid does this by looking at a variety of properties on the child elements, such as their alignment, and then places them accordingly. The buttons defined in this example are positioned 10 pixels from the left, and 10 and 40 pixels from the top edge of the grid respectively. Because the buttons are top- and left-aligned, the grid asks the buttons for their desired size and sets the height and width of those elements accordingly (which means that they will have just enough room to accommodate their contents). Had you set the horizontal and vertical alignments differently, the outcome would have been different as well. For instance, if the horizontal alignment were set to stretch, the button would stretch all the way across the grid with 0 pixel space on the right (because the right margin is set to 0).

Oh, and by the way, the buttons themselves were able to indicate their space requirement simply because their content is a sub-element (which happens to be a string in this case) and which the button lays out. In other words: The button is also a layout container (capable of holding a single element, just like the window object). Any XAML UI can really be thought of as a giant layout hierarchy.

It doesn't stop there. Try replacing the grid element in the above example with a StackPanel. This container arranges the buttons in a vertical top-to-bottom stack. Change the Orientation property on the StackPanel to Horizontal, and the stack arranges the individual buttons left to right, rather than top to bottom. Change the StackPanel to a WrapPanel, and you end up with a flowing layout, where each button is flown in left-to-right until there's no more horizontal space and it flows to the next line, much like document layout, or the layout on an HTML page.

But things get better! Consider a TabControl, for instance. This control is merely a layout container that contains a collection of TabItems (individual pages). The TabControl takes the Header elements from the contained TabItem elements and shows them across the top one after the other (creating a "tabbed dialog with multiple pages" appearance). The TabControl also chooses to show the selected page in the remaining main space, hiding all other child elements, until a different page is selected and the layout is refreshed. (Note: This is the default appearance and behavior. It's possible not only to move the headers to different locations, but to completely change the appearance of a tab control by replacing its layout logic through a mechanism called "items panel templating." But that's a different story and not the focus of this article). Each TabItem in turn contains—you guessed it—a single main element that's given the entire space assigned to the tab item content by the TabControl. It's common to use a layout panel (such as a Grid) as the TabItem's content, which allows putting any number of elements into each page of a tab control. At this point, the whole story starts over and you can put arbitrary elements into the container and continue this whole hierarchy without limitation.

I find the TabControl/TabItem example particularly interesting because it highlights that "layout" can be more sophisticated than just statically putting elements at a certain position on the screen. TabControls handle both headers and content elements. They also arbitrarily hide and show different UI elements as needed. Layout combines appearance with behavior in very powerful ways.

> Many developers don't realize it, but this approach is what makes XAML so exceptionally powerful.

Many developers don't realize it, but this very approach is what makes XAML so exceptionally powerful. Developers can create their own layout elements. In my XAML productivity article (see above), I showed how to create custom layout containers that can lay out entire data entry dialogs and more.

Breaking the Layout Accidently

XAML layout drives home the point that with great power comes great responsibility. Crafty and knowledgeable XAML developers can use these techniques to program circles around other technologies. Unfortunately, I also often see developers who're thoroughly confused by the layout features. Or, to be more exact, they're unaware that there is such a thing as a "layout engine". They simply assume that they should manually place elements on the screen. They tend to be befuddled by "this odd grid element in the window," and they end up with strange layout behavior that's inexplicable to them and often only surfaces during testing (or later). This is unfortunate, because understanding the layout system isn't rocket science once you're aware that a layout system exists.

There are many different ways to get layout wrong in XAML. A very common problem is related to horizontal and vertical alignment settings, especially when visual designers are used rather than coding XAML by hand. Keep in mind that many XAML layout containers (such as the Grid) use these settings to determine where to anchor the element. A horizontal alignment setting (HorizontalAlignment property) of "Left" usually positions the element relative to the left edge of the container, with the distance from the edge determined by the left margin. The width of the control is then based either on the Width property of the element, or, when the width isn't set, is auto-determined based on the contents of the control (such as the space requirements of the caption of a button). Whatever space remains to the right of the control is thus not part of the equation. There may be a visual space on the right side of the control, or there may not be. In fact, if the control is too wide to fit, it will just "hang out" of the right side of the container (and likely be cut off, although some containers allow keeping even overhanging elements visible). The right margin is completely ignored in this case.

Note that setting the horizontal alignment to **stretch** changes the layout entirely! With a stretch-setting, the

container doesn't even try to auto-determine the width of the element and instead sizes it to a certain margin. If the right margin is set to 10 pixels, for instance, the width of the element is always calculated so that it results in stretching all the way across to 10 pixels shy of the right edge of the container. This is also true when the user resizes the UI. The layout simply updates and keeps stretching the element across to keep the margin constant by adjusting the width of the control.

Now imagine the following scenario: A developer creates a data-entry UI with many text boxes and similar controls. Great care is taken to position all the elements so that they have the same width and their edges align. However, when the application is run and the UI is resized, some UI elements stretch and others stay in place, causing complete havoc and broken user interfaces. What happened? Most likely, the different elements had different alignment settings! An element that's set to appear 200 pixels wide may appear so because it's left-aligned and has a width of 200 pixels, or it may be stretch-aligned with a right-margin that's perfect to make it 200 pixels wide in the designer. When the app is executed and the container sizes change, the left-aligned controls remain at their widths, and the stretch-aligned controls grow and shrink, possibly overlaying other controls or disappearing completely. I see this problem happening regularly in real-world applications because it's easy to accidently "anchor" elements to the right edge in visual designers (which really merely sets the alignment). Some UI design tools even try to make smart guesses as to where to anchor controls (often causing two very similar controls to have different alignment settings). Luckily, the overall problem is easily fixed. Simply check the alignment settings to make sure they are consistent (both horizontally and vertically).

It's also interesting to note that explicitly specified width settings are considered even in stretch-scenarios, but can result in very odd outcomes. Let's say you have a container that's 500 pixels wide and the child control has a left and right margin of 50 pixels. This leaves 400 pixels for the element. However, if the width is forced to be 200 pixels, the element is actually centered in the 400-pixel space and thus adds an extra 100 pixels of space on each side. If the forced width is wider than the available space (let's say the width is 1000 pixels), it will be left-aligned within the available 400 pixels, and the 600 pixels that don't fit are clipped away. Needless to say, all of this results in some headscratchers on the side of the uninitiated. The lesson here is that explicitly setting the width on an element that is set to stretch horizontally is an anti-pattern. (The same goes for vertical stretch objects and the height setting).

These are just a few examples and each layout container exhibits different pitfalls. Most of them can be avoided simply by checking that settings are consistent across controls. It's also generally a good idea to verify that only those settings you really need are in the XAML code. UI design tools are notorious for adding lots of XAML code that isn't really needed. For instance, there's no reason to have a width setting if you intend to stretch-align the element horizontally. Even if things looked fine in design view, the setting isn't needed and will only cause problems down the road. Simply remove it from the XAML.

Another example is margin settings. If you left-align an element, there shouldn't be a right margin set. It may not appear that setting the right margin has any impact, but you may be in for a surprise when the user resizes the UI. The best thing to do is set the right margin to 0, or, if you're familiar with the impact of setting the margin (having the control cut off, but not properly resized, before it overlaps the right edge) you should deliberately set it rather than going with some accidental or old value that might have been added to the XAML by some other means.

Wrong Layout Containers

Misunderstanding the layout system often leads to incorrect fixes of perceived problems. I can't even begin to enumerate all the possible scenarios that could lead to such problems, but there's one scenario in particular that stands out as a classic problem in XAML UIs: Imagine a UI into which the developer intends to place a ListBox with several items:

```
<StackPanel Margin="10">
  <Label>Items:</Label>
  <ListBox>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
  </ListBox>
</StackPanel>
```

Apparently, the developer intends to display a label as a sort of heading for a list of items, followed by the list itself. The list is populated with manually defined items in this example, but the control could also be data-bound and thus have an unpredictable number of items. All of this is achieved by putting the label and the list into a stack panel, which indeed arranges these elements top-to-bottom, as intended (with a nice margin around it all).

All of this looks fine and works as intended. However, problems start when the user resizes the UI to a point where it isn't tall enough to fit the entire list of items. At that point, you'd expect the list to show a scroll bar so the user can scroll up and down to see more items. However, this isn't what happens. Instead, the list is cut off and no vertical scroll bar appears. Even more confusingly, the horizontal scroll bar works just fine whenever the list gets too narrow to show all the items. What's going on here?!?

The answer lies in the chosen layout elements and their behavior. Let's examine the elements one by one. The stack panel has no settings other than the margin, which means everything else is using the defaults (unless there's a style in place that changes those defaults). So the question is whether the complete stack panel is sized correctly. How can you find out? A poor-man's debugging tool is to set the element's background color to a color that stands out (like red). This way, you can see exactly how much space the control occupies during runtime. However, depending on the parent element where this stack panel resides (such as a window or a grid), you're likely to discover that the element's size is exactly as you would anticipate. So that's not it.

CODE Framework

CODE Framework is CODE Magazine's companion framework for developing advanced business applications. Among other features, CODE Framework supports various kinds of XAML development (with a particularly rich set of features for WPF applications). CODE Framework provides many components that help avoiding layout anti-patterns. The framework is open-source and completely free of charge and it can be used either as a complete framework, or simply by moving individual elements (such as layout objects) into other projects. For more information, visit www.codemag.com/framework

The next step is to take a look at the child elements and at the layout behavior of the container. In the case of a stack panel with vertical orientation (which is the default orientation of stack panel contents), the layout algorithm takes each child and looks at the element's height. It then places the elements top-to-bottom based on the element's desired height, and the width is set to use the entire available width within the stack panel. So the label goes at the top and the list goes in next. The important part is that the list is given the height it asks for. If there are just a few items, that height may be 50 pixels. If there are a thousand items, the height may be 20,000 pixels. Whatever that height is, the stack panel grants that height to the list, *with the overlap simply hanging out the bottom* and most likely being visually clipped away so it's invisible (but still there). And that's exactly where the problem lies. The list box has the capability to create a scroll bar whenever there isn't enough vertical space to fit all the items, but since the stack panel grants the list what amounts to infinite vertical space, the list is never clued in to the fact that items may not be visible.

> Even Internet research can produce bizarre suggestions and doesn't even come close to fixing the problem.

What's the solution to this issue? Well, that is where the problems really start for many developers. Even some Internet research can produce bizarre suggestions. One such suggestion is to set the height of the list box to a specific value (such as Height="150"). This way, the list is always exactly 150 pixels tall, and that's the height the stack panel will allow. When there are more items than fit into 150 vertical pixels, a scroll bar appears. However, this is unacceptably awful because the list won't grow and shrink as the user resizes the UI. Instead, it always remains at 150 pixels. It allows you to scroll through thousands of items within that 150 pixel space, but it will never grow larger than that. Worse, it will never be smaller either. So if the user resizes the UI too small, the list will be cut off at the bottom. I see a lot of people resorting to this approach in desperation. Granted, it may be preferable over no scroll bar at all, but it's nowhere near what a professional UI should be like. As a general suggestion, it's usually a bad idea to start interfering with automatic layout by forcing hardcoded dimensions for specific scenarios.

Another approach I see suggested regularly on the Internet is to wrap the entire thing into a ScrollViewer:

```
<ScrollViewer Margin="10">
  <StackPanel>
    <Label>Items:</Label>
    <ListBox>
      <Label>List Box Item</Label>
      <Label>List Box Item</Label>
      <Label>List Box Item</Label>
      <Label>List Box Item</Label>
      <Label>List Box Item</Label>
    </ListBox>
  </StackPanel>
</ScrollViewer>
```

A scroll viewer is another layout container with a special trick up its sleeve. It allows its members practically infinite space. If these child elements use up more space than is available, scroll bars are displayed, allowing the user to slide the entire contents of the scroll viewer up and down as needed (and left/right as well). This means that no matter how large or small the list gets, the scroll viewer always allows the user to scroll up and down to see all the items.

This is another awful solution for a number of reasons. For one, this doesn't just scroll the list; it scrolls all of the contents, including the label. Therefore, when the user scrolls down, the label disappears out the top. But this is probably the least of its problems. The main issue is that the scroll viewer doesn't fix the original problem, which is that the list doesn't provide scrolling when needed. Instead, the list never scrolls but simply slides up and down in its entirety, which is very different. Depending on the exact style, it may create odd visual effects with the border. It also means that in large lists, the list box control must always render all items which, as you probably guessed, it does by laying out all its elements using a vertical stack layout. A list with a million items will have to lay out and render a million items so the scroll viewer can then slide the whole thing up and down. It is very inefficient, and completely disables the list's ability to manage large item numbers using "virtualization.

Another issue is that you never see the scroll bar associated with the list box. Instead, you only see the scroll bar provided by the scroll viewer. This matters, because it means that whatever you do to change the list's scroll appearance and behavior never takes effect. For instance, if you create (or download) a style that specifically changes scroll bar appearance within lists, that won't apply to this hack, since it doesn't use its own scroll bar. Similarly, if there's special touch-behavior for lists that's different from general scroll viewers, your list won't get that benefit. You'll probably also experience problems with built-in list behavior, such as programmatically making items visible (depending on the exact elements and styles used). In short, it's fair to say that your list stopped behaving like a list, which is bad.

> I've never seen a scenario where adding additional layout elements into the visual tree solves the underlying problem.

For this scenario, a general rule of thumb is that it's bad to introduce additional containers into the visual composition in an attempt to force behavior that you know is probably there by default, but doesn't show up for some odd reason. A control that has the ability to scroll should never require adding a scroll container. I've never seen a scenario where adding additional layout elements into the visual tree solves the underlying problem.

Super-Productive XAML Development

The author of this article previously wrote two articles on productive XAML development. One was about creating advanced layouts productively (Quick ID: 1011071), and the other was about creating lists of data efficiently (Quick ID: 112091). These articles have become timeless classics and are among our most-read XAML articles, discussing core fundamentals that remain as relevant today as they were when the articles were originally published.

REAL HEROES SOLVE REAL CHALLENGES

OPENAIR 2015

THE WEB ACCESSIBILITY CHALLENGE

Compete in a global web accessibility challenge that pairs web professionals with nonprofits looking to create a website. Be trained by the leading experts in accessibility. And don't worry, there's no cape required. Tights optional.

October 1, 2015

Register now at
www.air-rallies.org

/Knowbility

@knowbility

knowbility

The right approach is to fix the root problem, which is that the chosen layout was not the right tool for the job. Although it created a vertical arrangement of the label and list, it didn't size the list correctly. Therefore, a different container is needed and there are several choices. One is to create a custom layout container. For instance, there's a special variation on the stack panel idea that allows you to specify a stack with the last item in the stack filling all remaining space by forcing it to a certain height. (You can download this element—called **BidirectionalStackPanel**—as part of the free and open-source CODE Framework, here: www.codemag.com/framework.) For this specific example, you can even use a grid to achieve the desired result:

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinitions Height="Auto">
    <RowDefinitions Height="*">
  </Grid.RowDefinitions>
  <Label>Items:</Label>
  <ListBox Grid.Row="1">
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
  </ListBox>
</StackPanel>
```

In this scenario, you define a grid with two rows. The first one auto-sizes its height to whatever the content requires (the height of the label, in this case). The second row uses up the remaining space, which means that it will occupy all of the remaining vertical space. The list box is assigned to that second row and uses up all the space in that row, which is exactly what you want. Because the list is forced to take on the size of the second row, it knows how much vertical space it has been assigned, and can thus show a scroll bar as needed and also manage the number of items. If there's only enough space to show 10 items, it only shows and lays out those 10 items, rather than a million.

Overall, this approach is quite different when compared to the previous fixes. You've now chosen a layout that positions elements correctly, rather than trying to compensate for incorrect layout.

On a side note: It's possible to achieve something very similar with a grid without having to create rows. You could have set a top margin on the list to accommodate the label above, like this:

```
<Grid>
  <Label>Items:</Label>
  <ListBox Margin="0,30,0,0">
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
  </ListBox>
</StackPanel>
```

Although I wouldn't consider this to be completely broken, I still consider it much inferior to the previous approach. The reason is that the row-based approach is capable of auto-adjusting the space allocated to the label. This allows changing the label's style to use different fonts or font sizes. It allows for scaling and zooming. It makes internationalization easier. It's generally more functional and maintainable, which is why I'd choose that approach every time. The more layout the layout engine can do automatically, the better. I admit that the more verbose nature of the row-based example is annoying, but there are solutions to that issue as well. For instance, in my recent "XAML Magic: Attached Properties" article (http://www.codemag.com/Article/1405061), I showed how to create an attached property that allows for this syntax:

```
<Grid ex:Ex.RowHeights="Auto,*">
  <Label>Items:</Label>
  <ListBox Grid.Row="1">
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
    <Label>List Box Item</Label>
  </ListBox>
</StackPanel>
```

With this, the two-rows version of the grid is similar in lines-of-code (or characters-of-code) to the one-row-with-margins version, but the two-row version is more advanced, and arguably less error-prone to create.

Manual Layout versus Automatic Layout

Generally speaking, not understanding the potential of automatic layout in XAML is one of the biggest issues I see in XAML apps. It's not so much that it's erroneous to manually put elements and controls onto hardcoded screen positions, but it's simply missing out on one of the most powerful features. It's a bit like using SQL Server but not creating relationally structured data. My most recent example with the grid's two rows automatically positioning elements barely begins to scratch the surface of what's possible. Things get really interesting once custom layout elements come into play.

A complete discussion of creating custom layout elements is beyond the scope of this article (and, as I've mentioned several times, I covered in depth in that earlier article). The short version is that you can create your own layout element by subclassing the Panel class and by then performing two tasks:

1. Measure the elements in the screen.
2. Arrange those elements on the display surface.

This is done by overriding the MeasureOverride() and ArrangeOverride() methods respectively. The beefy part of the operation is in the arrange step, which iterates through all the children of a panel and assigns them each their own rectangular space on the screen.

Fundamentally, creating arrange logic is a straightforward task. Envision what the StackPanel class would have to do, for instance. It starts at position 0,0 and assigns the first element its own rectangle that starts at that position

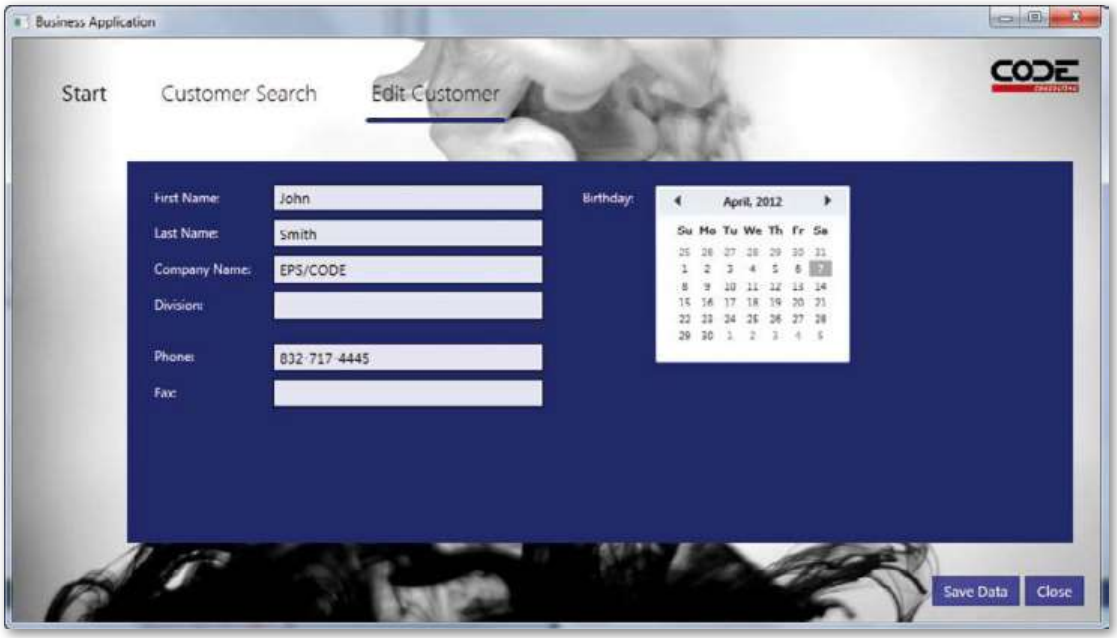


Figure 1: An automatically laid out data edit form created with only a few lines of XAML code (see Listing 1)

Listing 1: The XAML code needed to create the result shown in Figure 1.

```
<l:View
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:l="clr-namespace:MyLayout">

    <TextBlock>First Name:</TextBlock>
    <TextBox Text="{Binding FirstName}" />
    <TextBlock>Last Name:</TextBlock>
    <TextBox Text="{Binding LastName}" />
    <TextBlock>Company:</TextBlock>
    <TextBox Text="{Binding Company}" />
    <TextBlock>Position:</TextBlock>
    <TextBox Text="{Binding Position}" />

    <TextBlock Margin="0,25,0,0">Phone:</TextBlock>
    <TextBox Text="{Binding Phone}" />
    <TextBlock>Email:</TextBlock>
    <TextBox Text="{Binding Email}" />
</l:View>
```

and has the width of the entire panel. The height is the height the element desires, which was previously determined by measuring the control. The same process is then repeated for the next element, except it's moved further down vertically. This works the same for all other layout elements, except that the actual logic for arranging elements is different in order to arrive at different locations.

One example I like a lot is the creation of a layout panel that can arrange data edit forms. **Figure 1** shows such a user interface (the one I show how to create in that referenced article). The idea is to organize the children of the panel into pairs so that they always have a label and a control, and then to arrange them in columns and rows to arrive at the result shown in **Figure 1**. You can add some special features, such as group breaks or multiple columns. In fact, in real-world implementations, such layout panels will have many more features to account for special cases and more sophisticated layout, but the underlying ideas remain the same. (Note: If you are looking for different implementations of standard layout elements, you can download the free CODE Framework from www.codemag.com/framework, which includes many such layout panels).

Listing 1 shows the XAML code associated with the UI shown in **Figure 1**. The interesting part of **Listing 1** is not so much what is there, but what isn't there. Most people expect significantly more code to achieve what's shown. In fact, in many cases, I see hundreds or even thousands of lines of XAML code, when I could achieve the same result with just a handful of lines (another anti-pattern in itself). Clearly, it's more desirable to achieve the same result with much less code. But it goes a bit beyond that. Not only is there an advantage in writing less code, but the smaller code base is also more maintainable and flexible for future changes. It also often performs better. Therefore, it's better all around, and missing out on this advantage is an anti-pattern.

Conclusion

The layout systems found in all XAML technologies are one of the quintessentially unique features that set XAML apart from other environments. Yes, some of these things can also be achieved in other environments, but a lot of extra work has to be done; in XAML, it's just built in at the core. Missing out on any of these features, or perhaps disabling them by misunderstanding the fundamentals, is the cardinal sin of XAML development and the root cause of XAML projects that grow into slow and unmanageable monsters, an anti-pattern of the worst kind.

Markus Egger

Azure Skyline: Hello World—Virtual Machines in the Cloud

Cloud development is the hottest job skill around. The cloud phenomenon is undeniable. As a part-time curmudgeon, I wondered if I'd be writing about the cloud as a passing fad by now, but quite the contrary. I think the cloud is the greatest thing to come along since the debugger. The cloud landscape is starting to take shape and the dust is starting to settle just a little bit.

Mike Yeager

Mike is the CEO of EPS' Houston office and a skilled .NET developer. Mike excels at evaluating business requirements and turning them into results from development teams. He's been the Project Lead on many projects at EPS and promotes the use of modern best practices such as the Agile development paradigm, use of design patterns and test-drive and test-first development. Before coming to EPS Mike was a business owner developing a high-profile software business in the leisure industry. He grew the business from 2 employees to over 30 before selling the company and looking for new challenges. Implementation experience includes: .NET, SQL Server, Windows Azure, Microsoft Surface and Visual FoxPro.

So don't worry if you're new to all this. You're not too late; some of the kinks have already been worked out and it's a lot of fun. To get you off to a good start, you need to get signed up with an Azure account and subscription and then create your first virtual machine (VM). Creating VMs is one of the original features of all clouds (hosting websites is the other), and hosting VMs is still one of the most common uses of the cloud. It's essentially the "Hello World" of cloud skills.

Get Signed Up for Azure

If you don't already have an Azure account, type **azure.com** into your browser (you'll be redirected to **azure.microsoft.com**) and look for the words "Free Trial." Those words have been on that page almost since Azure began. At the bottom of the page, you'll notice that if you have an MSDN subscription, or if you're a new business startup or a student, you may qualify for ongoing free Azure credits. For now, let's just pick the Free Trial subscription and you can add one of the other programs to your account later if they apply to you.

You need a Microsoft account to create your Azure account, and if you don't have one already, you can create one right here. By adding the Free Trial subscription to your account, you'll get a free month of Azure with up to US$200 in free credit, so this won't cost you a thing. You'll have to provide a credit card number to create the subscription even though it won't be charged. Once logged into the Azure portal with your account, you'll need to create an Azure subscription, as shown in **Figure 1**.

You'll be the administrator of your Azure account, so you'll have carte-blanche. You can add co-administrators or give out access to specific items, but soon, there'll be a rich role-based security mechanism allowing you much

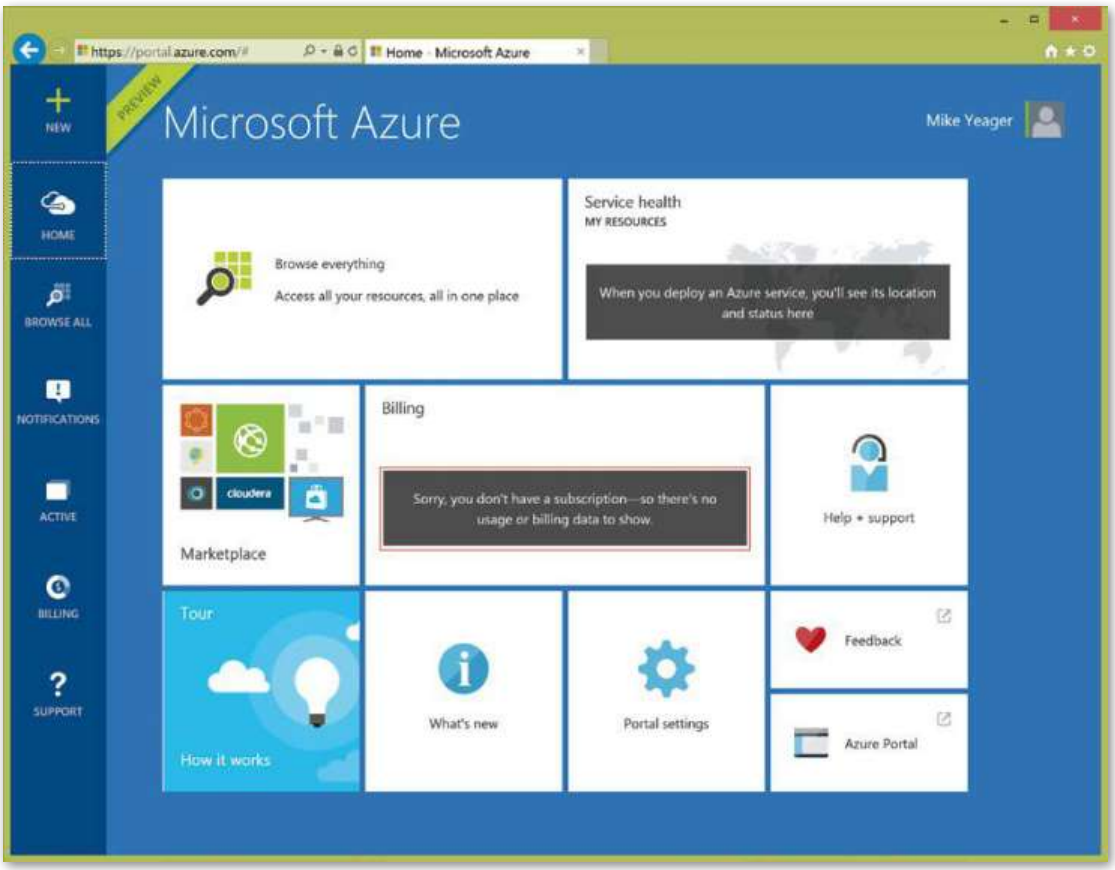


Figure 1: Sign up for an Azure account by logging into azure.com with your Microsoft account.

Learn How To Master Big Data

BigData TECHCON

November 2-4, 2015
CHICAGO

Holiday Inn Chicago Mart Plaza River North

Choose from 55+ classes and tutorials!

Attend Big Data TechCon to get practical training on Hadoop, Spark, YARN, R, HBase, Hive, Predictive Analytics, and much more!

Take a Big Data analytics tutorial, dive deep into machine learning and NoSQL, learn how to master MongoDB and Cassandra, discover best practices for using graph databases such as Neo4j and more. You'll get the best Big Data training at Big Data TechCon!

www.BigDataTechCon.com

People are talking about BigData TechCon!

Great for quickly coming up to speed in the big data landscape.
—Ben Pollitt, Database Enginee, General Electric

There was a large quantity and variety of educational talks with very few sales lectures. It was just informative and inspiring. This was the best conference ever! Get a ticket for 2015!
—Byron Dover, Big Data Engineer, Rubicon Project

A BZ Media Event

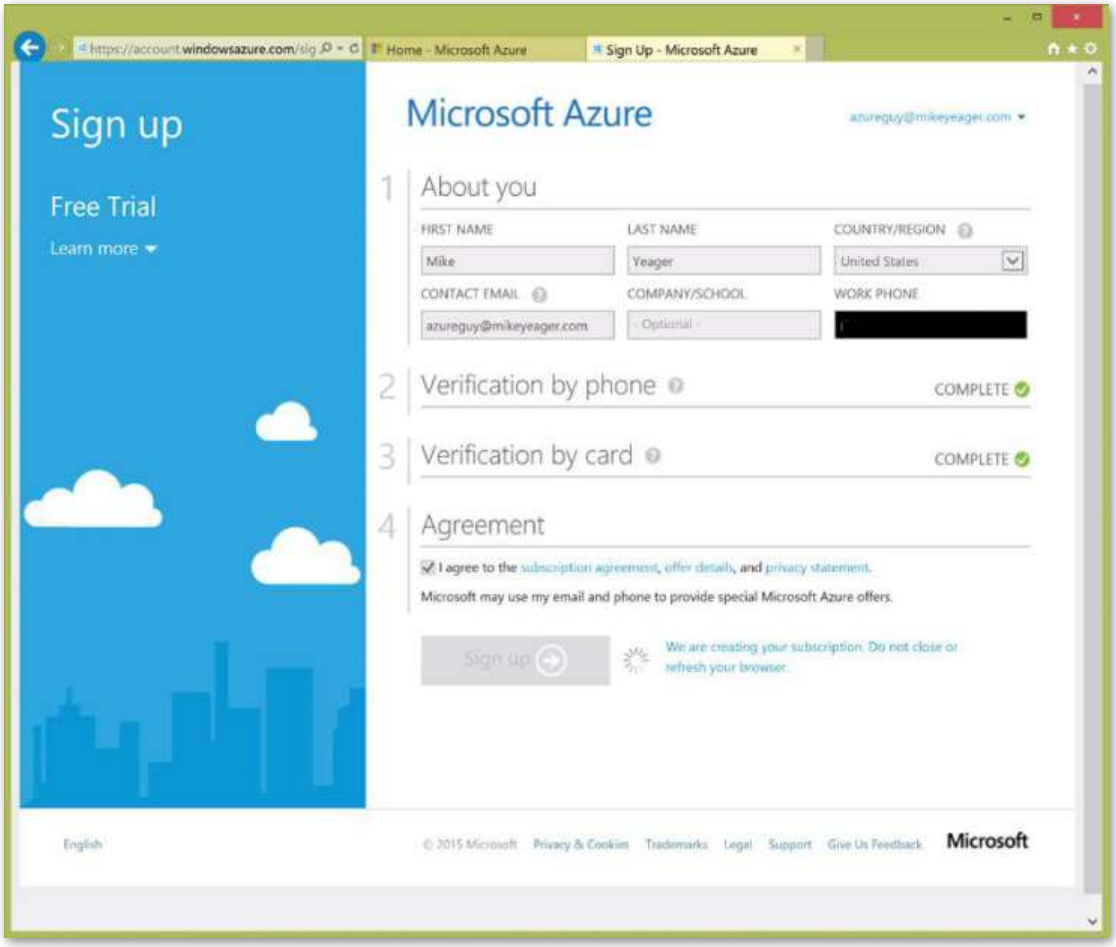


Figure 2: After logging into Azure, create a subscription.

better control of your Azure account. Also note that your login can be linked to multiple subscriptions at once, so you can have your own account, as shown in **Figure 2**, and be part of your company's account and others, all with the same login.

Virtual Machines

If you've used third-party virtualization products such as VMware, VirtualBox, or Parallels (on the Mac) or you've used Hyper-V (Windows 8/Windows Server 2008 and later) or the older Virtual PC (Windows XP/Windows 7), you're already familiar with the idea of VMs. Instead of installing Windows on a dedicated piece of hardware, it's installed it into a portable sandbox. Many VM sandboxes can run on the same hardware as your production projects and share that hardware, cutting hardware costs. As far as your OS and software installation are concerned, they don't know if they're installed directly on the hardware and have it all to themselves or if they're installed in a VM sandbox, sharing hardware with other VMs.

There are usually two parts to a VM: a virtual hard disk and a configuration file. You're probably familiar with the .vhd or .vhdx file formats for virtual hard disks (Azure uses the .vhd format). They represent an entire hard disk volume inside of a single file. In recent versions of Windows, you can mount these files and they show up and can be used like any other hard drive. In most cases, an entire bootable operating system is loaded onto that hard drive. Other software, such as Microsoft Office, DropBox, or your custom apps can then be loaded onto the virtual drive as well. As long as the drive isn't currently mounted, it can be moved and copied like any other file.

In order to boot a VM from the .vhd file, there must be a configuration file on the host system. This configuration file is how the OS on the host computer configures the environment sandbox that the VM will run in. For example, the configuration file may say that your VM gets access to 4GB of RAM, two virtual processors, a (shared) network card, and maybe an extra disk drive (.vhd) in addition to the OS drive. It may also configure whether the VM and the host OS share the same clipboard, printers, and other features.

Why so much talk about how VMs work? Aside from explaining a bit about how your computer will run in Azure, one of the cool features of Azure is that it uses the same standards as Hyper-V and VMWare. That's important because it means you can move and copy your VMs between your own computers and Azure. You can't do THAT on Amazon Web Services!

The other cool thing about it? Since your virtual hard drives are just files, you can store them in Azure storage for almost nothing! Last I checked, the retail cost

Take your Android development skills to the next level!

AnDevCon

The Android Developer Conference

Dec. 1-3, 2015

Hyatt Regency Santa Clara

Register Early! AnDevCon Santa Clara will sell out!

Get the best Android developer training anywhere!

- Choose from more than 75 classes and in-depth tutorials
- Meet Google and Google Development Experts
- Network with speakers and other Android developers
- Check out more than 50 third-party vendors
- Women in Android Luncheon
- Panels and keynotes
- Receptions, ice cream, prizes and more (plus lots of coffee!)

Whether you're an enterprise developer, work for a commercial software company, or are driving your own startup, if you want to build Android apps, you need to attend AnDevCon!

Check out the program online at www.AnDevCon.com

Android is everywhere! But AnDevCon is where you should be!

A BZ Media Event #AnDevCon

to store files in Azure was about $0.02/GB per month. If your virtual hard disk is of the dynamically expanding type, it's only about the size of the files. Most empty space on the drive isn't saved in the file since it's allocated only when needed. So a fresh installation of Windows 8.1 with Office and Visual Studio installed is probably only about 16GB in size. That means it costs only about $0.32/month to store the virtual disk drive in the cloud when it's not running. If you just signed up for a free Azure trial, you have a free $200 to spend. Your credit card won't be charged if you accidentally go over unless you explicitly approve it, so have no fear! When it's running, you also pay for that time. More on that later.

Create Your First VM in Azure

In the previous section, I mentioned how you can load your own copy of an Operating System and software onto your VM. This is a great way to spin up a computer, but it's a pretty slow process. First you install the OS, then you patch the OS repeatedly until it's up to date, then you install all the software, then you patch the software repeatedly. It usually takes the better part of a day popping back and forth to the VM to keep things moving.

Luckily, the folks at Microsoft (and hundreds of other companies) have already done this for us for most scenarios. They've provided us with a gallery of pre-built, pre-patched virtual hard disks that are ready to fire up and go. Want to try out some test scenarios on SharePoint? Pick an image with Server 2012 R2 and SharePoint already installed and patched. Want to learn to work with Oracle? Pick an image with Oracle pre-installed and get right to work. And you're not limited to running Windows or the Microsoft stacks. There are images for Linux, PERL development, IBM WebSphere, and others ready to go. It's a great way to try new things without mucking up your main computer. In most scenarios, the cost of the pre-installed operating system and software on the VM is built into the cost of running the instance, so there are no up-front licensing fees.

> Selecting a data center might be one of the most important options you choose when creating a new VM.

Let's pick something interesting for this VM. How about the next version of Visual Studio or the next version of Windows? As I write this article, Visual Studio 2015 is a release candidate and Windows 10 is in preview and scheduled to be rolled out at the end of July. Both are offered as pre-built images in Azure. I'll pick VS 2015 for this demo, but you can pick anything that's of interest to you. Sign into the Preview portal (portal.azure.com), click the **New** icon that looks like a plus sign in the upper left-hand corner and choose **Compute** and then **Marketplace**. Or choose the full portal (manage.windowsazure.com) and choose "Virtual Machines." Choose the **New** icon that looks like a plus sign in the lower left-hand corner and choose **From Gallery**. Search for Visual Studio 2015. You'll

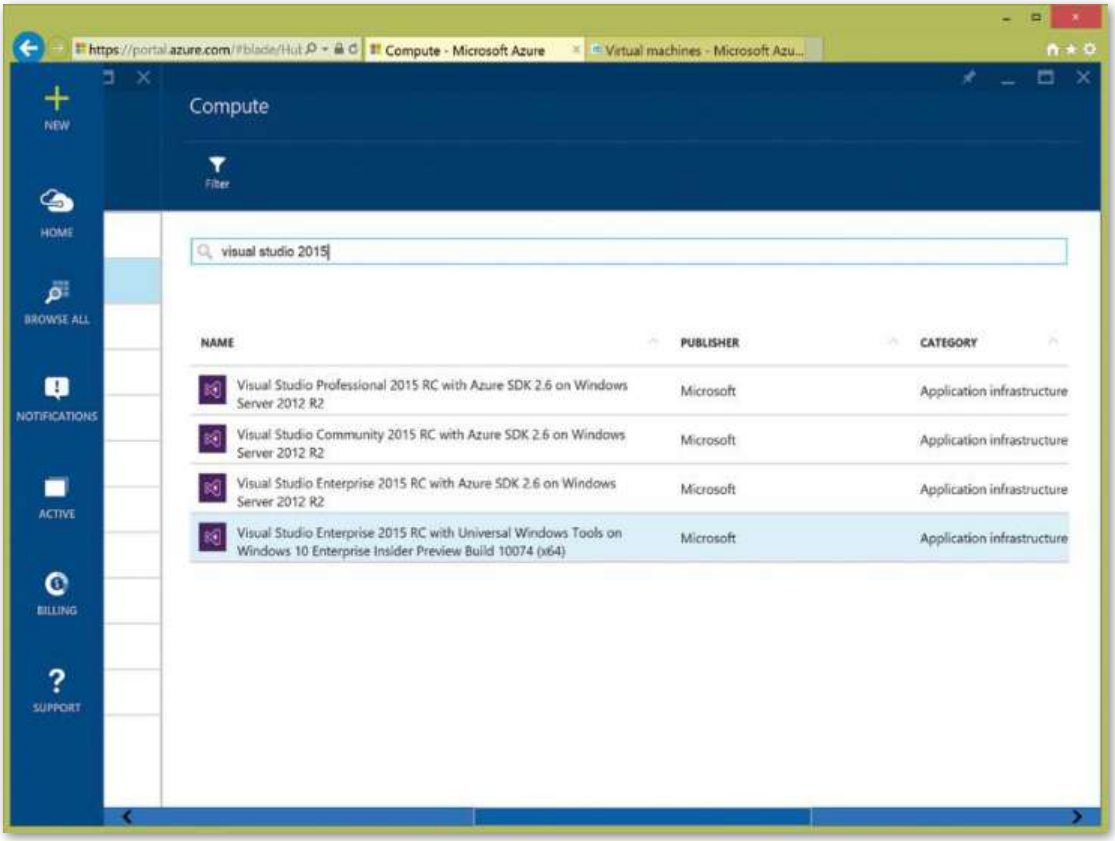


Figure 3: Choose from thousands of pre-built VM images.

Two Azure Portals

The original Azure portal is located at manage.windowsazure.com and it's the more complete of the two. A few years back, the Preview Portal was launched at portal.azure.com and it's been round ever since. It has a very modern look and feel, but I find that it can be a bit hard to navigate. Some things are easier to do in the preview portal, some things are easier to do in the original portal, and some things can only be done in one or the other, so unfortunately, there's no single right answer here. You'll probably end up using both, although like most of us, you'll probably spend more time in the original portal because it's more complete. At this point, I think it's safe to assume that the Preview moniker has outlived anyone's expectations and we'll probably be stuck with two portals for a long, long, time.

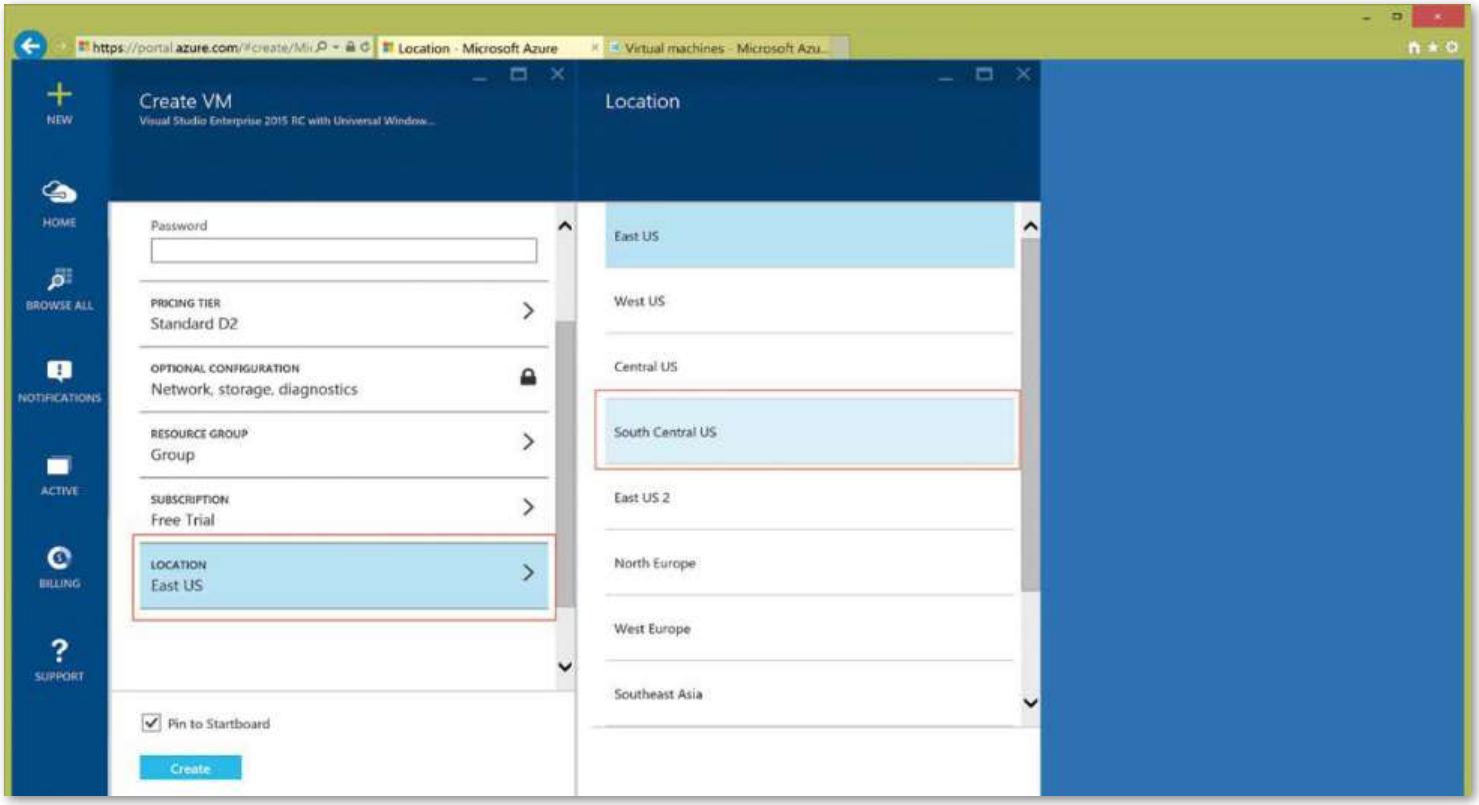


Figure 4: Choosing a Region is the most important choice to be made when creating resources in Azure.

find the latest release installed on the latest version of Windows Server; in this case, you can choose from Windows Server 2012 R2 or Windows 10 Enterprise, as shown in **Figure 3**. See the sidebar about why there are two Azure Web portals.

Now you have to configure your VM. The first step is usually to decide where you want your VM to run. This isn't the first step on the page, but it's a good habit to start with this step. Typically, you'll pick the Microsoft Azure data center region closest to you, as shown in **Figure 4**. In my case, I live in Houston, TX, so I'll pick the South Central US data center, which is near San Antonio, Texas, not far from me. As you do more and more in Azure, you'll have to start being a bit more careful about where you spin up your resources because resources that need to talk to one another work best when they're in the same data centers. It's also not a trivial task to move resources from one data center to another after they're created, which makes this the most important option you will make when creating a new VM. Moving VMs among regions will likely get easier in the future, but why make it hard on yourself? For what you're doing following along in this article, you can probably pick any data center in the world. The only difference you're likely to notice is that the farther away you are from the data center, the more latency you'll have in your connection.

Next, choose a name for your VM. I'll call this one **Win-10VS2015RC**. Enter a name and strong password for the local administrator account on the new VM. Note that Azure won't allow you to enter "Admin" or "Administrator" as the account name. It also requires a strong password and you can't use common passwords like "P@ssw0rd" because that just makes it way too easy for a hacker. I'm going to enter Myeager and Wx34!27a5. Finally, choose how much horsepower you want. There are a few different types of configurations to choose from, each optimized for different types of workloads. The A series is the most generic, least expensive, and will work well for most workloads. The D series is designed for workloads that are more disk-intensive and includes an SSD drive in addition to the OS drive. The G series is designed for high-compute loads and the DS series has the highest performance since the OS drive is built on solid state disks (SSDs). Currently, the DS series is only available in certain locations. For the purposes of this article, you can't go wrong. Within each series, you can choose just how much horsepower you want to pay for. A0 is the smallest configuration in the A series and currently gives you a quarter of a virtual CPU core, three-quarters of a GB of RAM, and a maximum 300 IOPS on a very small hard disk for about $13/month running full time. I'd like a bit more horsepower, so I'm going to choose the A3 Standard with four virtual CPU cores, 7GB of RAM and 100GB of hard disk with a max of 4k IOPS, as shown in **Figure 5**. This set up will cost me about $268/month running full time. I only expect to run it for about 10 or 15 hours, so I expect to burn about $4.50 of my $200 credit.

Start it up! You'll see that it takes several minutes to provision your new VM. During this time, Azure is making you a copy of the drive image you chose and storing it in Azure storage. This image includes both a .vhd file and a configuration file. It then modifies the configura-

tion to the VM series and type you want and registers that VM to your account. Once provisioning is complete, Azure starts up your VM. Windows 10 Enterprise configures itself for the hardware and resources allocated to it much like when you start up a new computer the very first time. Just like on a new computer, this can take several minutes. The standard disk drives in Azure are notoriously slow. If you're used to a nice fast SSD, they'll seem painfully slow, but once the computer is booted and Visual Studio is cached in memory, the standard drives will be plenty fast.

Take It for a Spin

Press **Connect** to start a Remote Desktop Session and connect to your new VM (see **Figure 6**). Don't forget to use the name and password you specified when you created the VM. Type a backslash in front of your login name to tell Windows to log into the local computer and not try to use an account on your home domain, workgroup, or computer. Once you log in the first time, Remote Desktop Services remembers your settings and you can connect to your VM directly without having to go through the Azure Web portal first.

Before You Go

Don't forget that you pay for VMs primarily based on the amount of time they run. You can shut down your VMs when you're not using them to save money. Once a VM is shut down, it loses its assigned IP address in Azure, but for the purpose of this article, you don't need to maintain the address.

VMs are relatively cheap to run compared to buying new computers, but the costs can add up quickly. Compared to our example storage cost of $0.32/month for 16GB, a beefy VM could cost hundreds of dollars if left running all 43,200 minutes in a month. Just running it eight hours a day, five days a week reduces that cost by almost 75%. And remember, if you're using a trial account, an MSDN account, BizSpark account, or educational account, Microsoft is giving you a big fat credit for free. Both of the Azure Web portals show you how much credit you have remaining on your account and how many days are left before the next cycle. I think the Preview portal does a nice job of displaying this right on the front page by default.

Azure billing can be a bit complex because each service can include multiple charges such as CPU time, storage space allocated, bandwidth used, etc. If you want a better idea of how you're being charged, you can see all of the details for charges by downloading a spreadsheet with all of the details. Click on the billing tile in the Preview portal and then look for the link on the right-

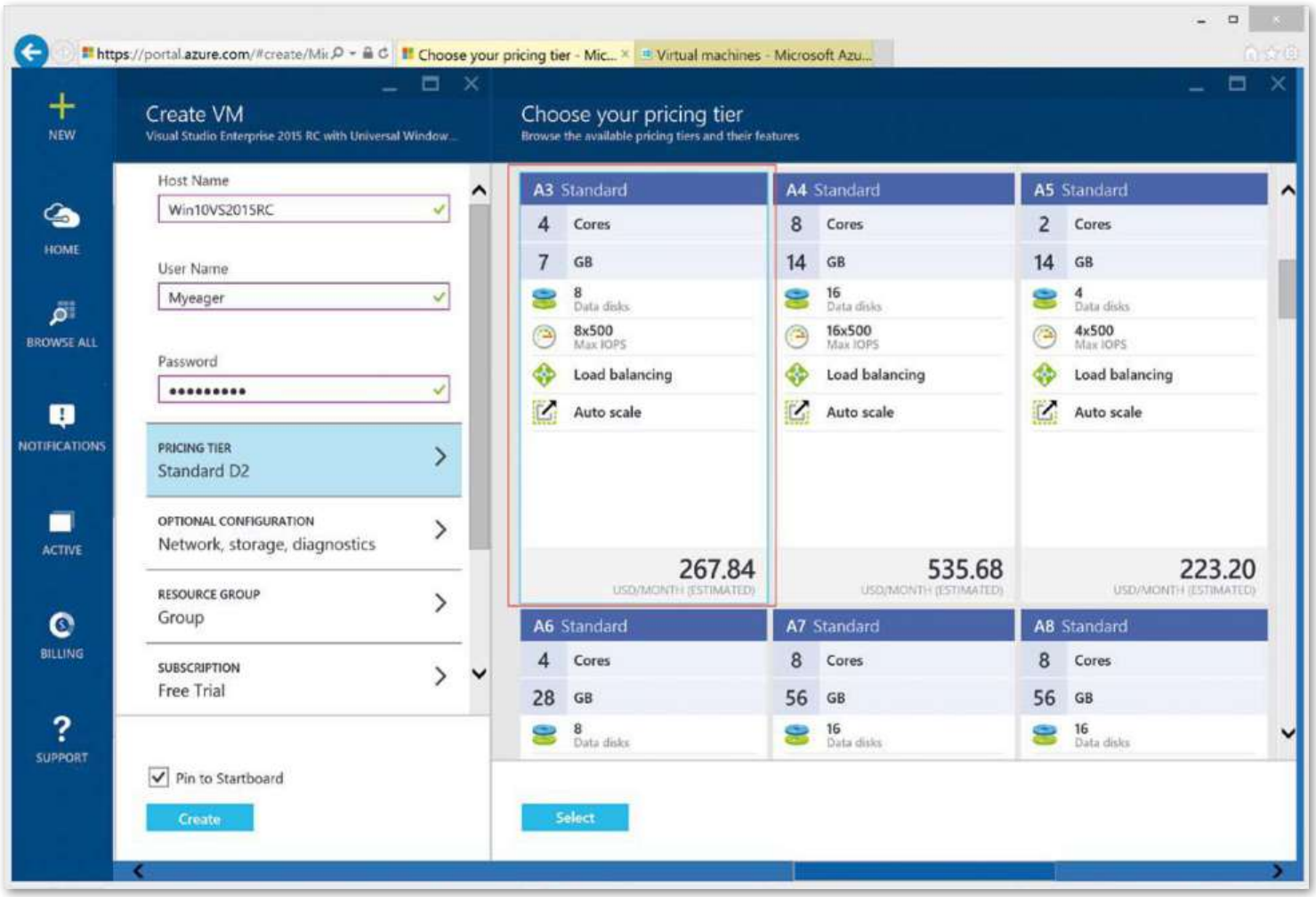


Figure 5: Choose how much horsepower you want to run and pay for.

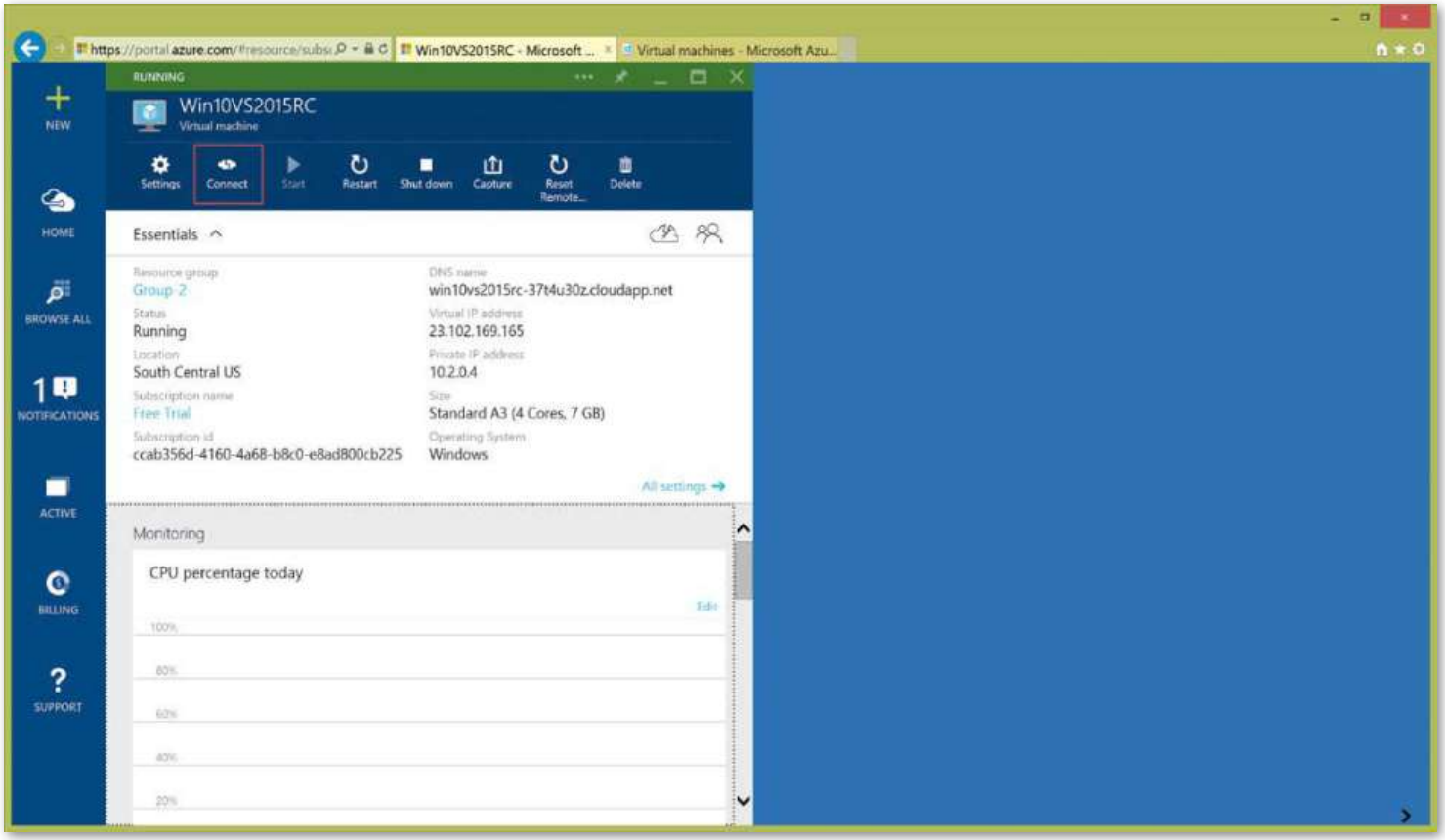


Figure 6: After connecting the first time, Remote Desktop will remember your connection settings.

hand side of the page. Your billing details are recorded, even if you're getting a credit, so this is a good way to see exactly what something will cost you in a production environment.

Once you're completely done with your VM, delete it from the portal. You'll be asked if you also want to delete the hard disks as well.

What's Next?

In this article, I just scratched the surface of Microsoft Azure. In fact, I've just scratched the surface of what can be done with VMs in Azure. Virtual networks can be created between on-premises and cloud networks, domains can be created in the cloud or extended to and/or from on-premises networks, and dedicated connections can be configured. Some of Azure's most unique features provide the ability to blur the lines between on-premises and the cloud and even to create large and complex data centers in the cloud, replacing the on-premises hardware of yesterday.

In addition to computers and networking, Azure covers a lot of other ground. As I write this, I count 46 distinct offerings in Azure, many with lots of variations and additions. Today, I covered the canonical example, the "Hello World" of clouds, creating a VM. In the coming issues, I'll talk about some of the other 45 offerings and I'm sure there will be more than 45 by the time the next issue of CODE Magazine hits the newsstand. I'll try to focus on those parts that are of interest to developers (as opposed to IT professionals).

I recommend taking a look around at some of the prebuilt images available in the Azure Marketplace. There are a lot of cool things you can try out and explore without having to spend days of setup and configuration. I could spend the next two months exploring what's available and learning about just a fraction of the software and systems available there.

Mike Yeager
CODE

Using Apache Hadoop on the Windows Platform

You've heard of Hadoop and the "circus" around big data; you've probably been interested enough to do a little research on the topic and you may be forgiven for thinking that this is only a toy in the Unix-like toy box.I'm happy to tell you that's not the case. Hadoop is alive and well on the Windows platform, both standalone via the Hortonworks Data Platform for Windows or,

Gary Short
gary@duncodin.it
http://www.duncodin.it
@garyshort

Gary Short is a freelance data science practitioner and trainer, based in Dundee, UK. He has a deep understanding of the full Hadoop and HDInsight environment, as well as an interest in Predictive Analytics, Social Network Analysis, Computational Linguistics, and Machine Vison.

if you prefer to use it as a service, you can do so through HDInsights on Azure. It's even available for you experiment with on your own development computers via the HDInsight emulator. The purpose of this article is to show you how to get up and running with the emulator so you can start extending your .NET skills into the big data arena.

Before I do that, however, I want to remove some of the confusion around the terms **MapReduce** and **Hadoop**. These two names seem to have become synonymous although in actual fact, they're very different. Let's look at them.

MapReduce

The best way to explain MapReduce is via an example. The canonical example for MapReduce is word count, so to stick with tradition, I'll make that the first example. Consider the following naïve algorithm for counting words in an input:

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Problem
{
  class Program
  {
  static void Main()
  {
    "one, two, two, three,
    three, three, four,
    four, four, four"
    .Split(',')
    .GroupBy(x => x)
    .ToList()
  .ForEach(group =>
    Console.WriteLine(
    "{0} appears {1} time(s)",
    group.Key.Trim(),
    group.Count()));
  }
 }
}
```

The algorithm takes the provided string, splits it on the comma character, groups by the words themselves, and then, for each group, prints out the word and the number of occurrences of that word in the group. The output from this algorithm can be seen in **Figure 1**.

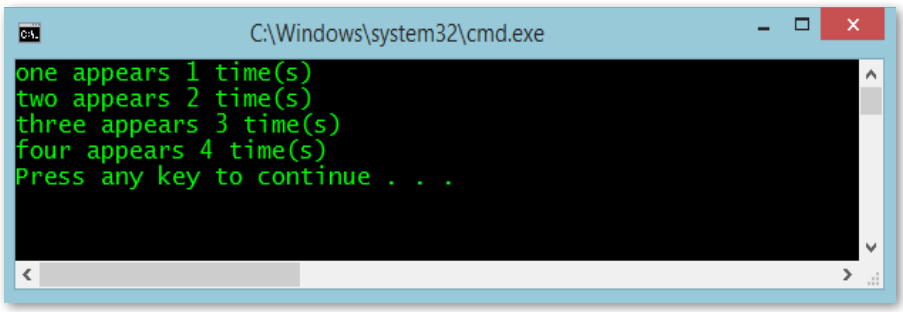


Figure 1: The word count results

This algorithm is constrained both by memory and by CPU. It's constrained by memory because as the size of the input string doubles, it reaches a point where it can no longer read the input directly into memory. You can mitigate that by reading the input one line at a time from a file. The algorithm is further constrained by CPU due to the input string continuing to double; it reaches a point where it can no longer calculate the word count in a timely manner.

> MapReduce is a scalability algorithm and Hadoop is a framework that supports that algorithm.

MapReduce is the algorithmic solution to this problem. By splitting the task into two functions: Map and Reduce. The Map function takes the input and maps it to a tuple output with a key and a value. In this case, the key is the word **appears** and the value is the integer **1**, signifying that the algorithm has found this word once. The Reduce function groups the output of the Map function by key and then reduces that output down to another tuple output. This time, the key is the **word**, and the value is the summation of the number of times that word has appeared. For example, if the output from the Map function is: ("the", 1), ("the", 1), ("the", 1), the output from the Reduce function is: ("the", 3).

Listing 1 provides an example of the earlier wordcount algorithm expressed as a MapReduce. You can see that the mapper splits the input and then emits a count every time it sees a word, and the reducer groups those words and performs a reduction by summation, giving the exact same result as the original algorithm, as can be seen in **Figure 2**.

Listing 1: The wordcount algorithm expressed as a MapReduce

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Solution
{
    class Program
    {
        static void Main()
        {
         Reduce(Map("one, two, two,
                    three, three, three," +
         "four, four, four, four"));
        }

        private static List<Tuple<string, int>> Map(string input)
        {
        var list = new List<Tuple<string, int>>();

        input
        .Split(',')
        .ToList()
        .ForEach(word =>
        list.Add(
        Tuple.Create<string, int>(word, 1)));

            return list;
        }

        private static void Reduce(
                            List<Tuple<string, int>> list)
        {
            list
                .GroupBy(tuple => tuple.Item1)
                .ToList()
                .ForEach(group =>
                    Console.WriteLine(
                    "{0} appears {1} time(s)",
                    group.Key.Trim(),
                    group.Count()));
        }
    }
}
```

The advantage of the MapReduce algorithm is that it allows the original algorithm to be split across a grid of commodity-compute nodes in a cluster, so now when the size of the input doubles, you can simply double the size of the cluster to maintain speed of processing.

Although you've solved the scalability issue, you now have other issues to deal with. For example, you're responsible for splitting the input data across data nodes, for collecting the output of the mappers and forwarding it to the reducer nodes, for dealing with hard drive failures in the cluster, and etc. That's where Hadoop comes in.

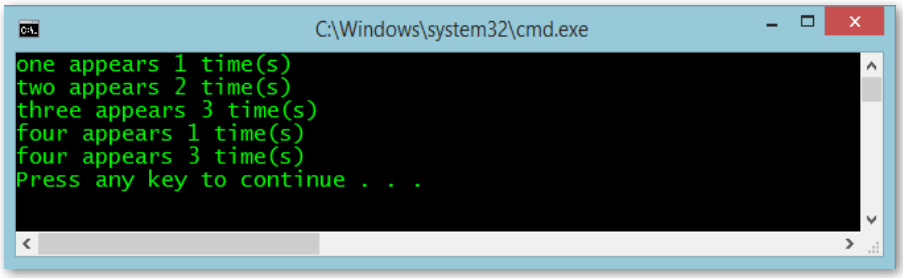


Figure 2: The algorithm returns the same results with MapReduce.

Hadoop

MapReduce is a scalability algorithm and Hadoop is a framework that supports that algorithm. The internal workings of Hadoop are outside the scope of this article but it's time to set up an instance of Hadoop on your development computer so you can start using your .NET skills in the big data world.

The easiest way to get started with Hadoop is via the HDInsight Emulator, which is installable via the Web Platform Installer. Launch the WPI and search for HDInsight, as shown in **Figure 3**.

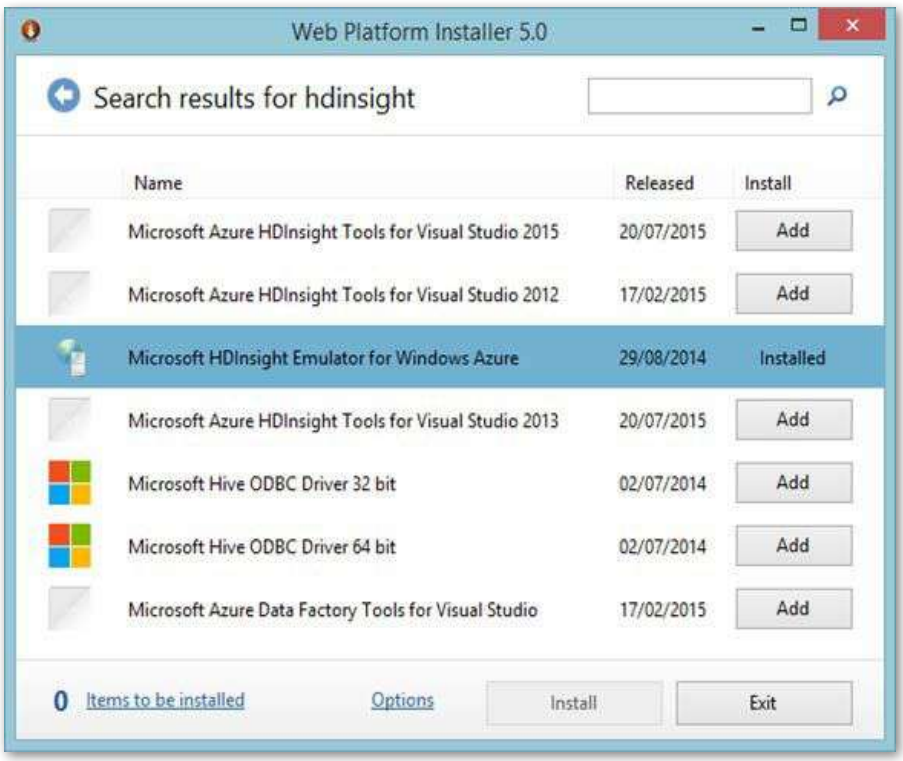


Figure 3: You'll need Microsoft HDInsight Emulator for Windows Azure.

You can see that I already have the emulator installed on my computer. If you search for the emulator and it's not found, that's Microsoft's subtle way of telling you that the emulator is not available for your platform, which is most likely due to the fact that you're on a 32bit computer and the emulator requires a 64bit computer.

The installer installs both the Hortonworks Data Platform for Windows (DPW) and the HDInsight emulator because HDInsight has a dependency on the DPW. It also installs and starts a number of services, indicated in **Figure 4**.

Any programming language that can read from and write to the console is allowed to define Hadoop jobs via its

streaming API. As C# is a language that meets the requirements, you can now go ahead and write your first Hadoop job.

Because I've already used a word count example, I'm going to change things up a little and use the example of predicting a horse racing result based on a specified input. The first thing to do is to create a new solution and add two console projects to it, one for the mapper and one for the reducer.

In this example, the Map function reads the horse, jockey, and course names from the console, fetches a bunch of stats relative to that combination, and then emits a key value pair where the key is the horse and the value is a list of stats connected to that horse. The input is fed one line at a time to the program via the console by the Hadoop framework. This is achieved by the code in the following snippet:

```
static void Main()
{
  string line = string.Empty;
  while ((line = Console.ReadLine()) != null)
  {
    string[] fields = line.Split(',');
   string horse = fields[0];
   string jockey = fields[1];
   string course = fields[2];

   string stats = GetStatsForHorseJockeyCourse(
            horse,
```

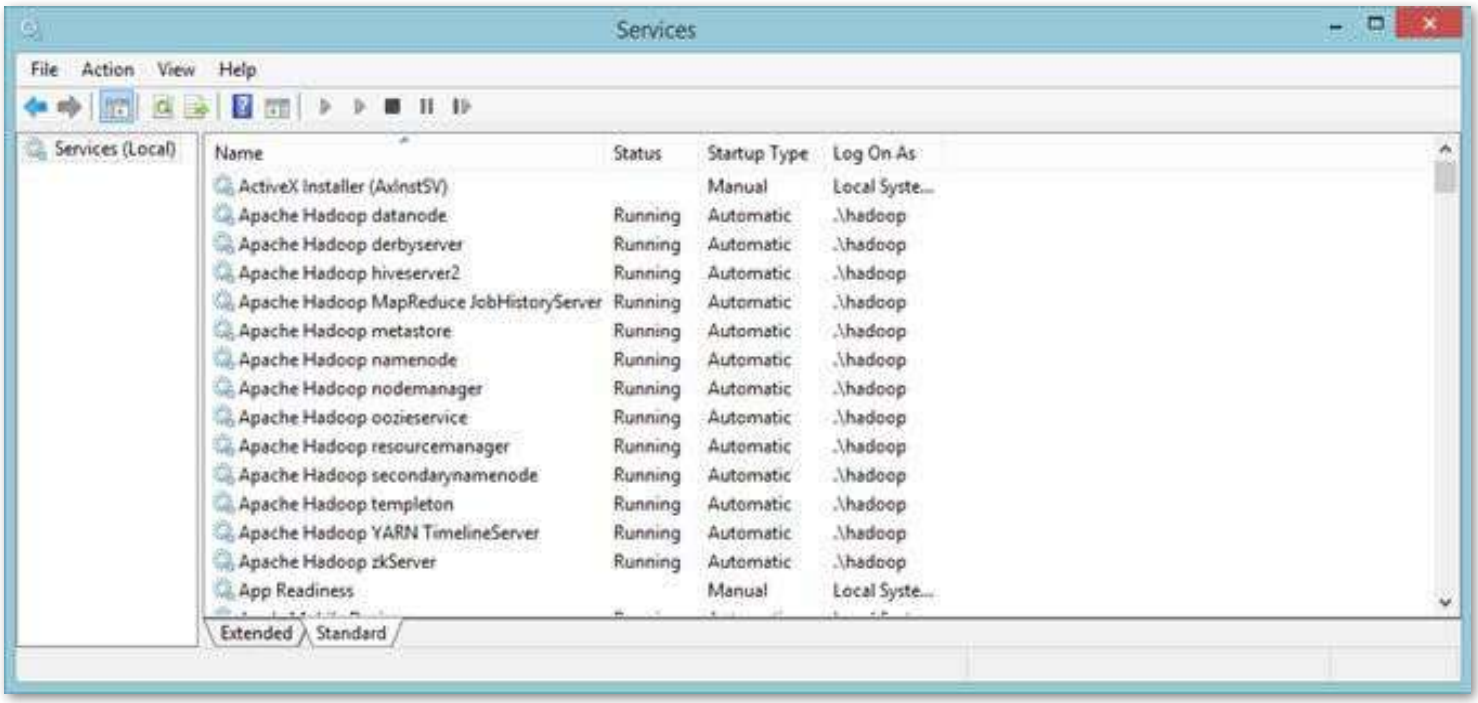


Figure 4: Some services are installed with the DPW.

Figure 5: This is the Hadoop fs –put command in action.

Figure 6: Pushing the Map function to Hadoop

Figure 7: Pushing the Reducer function to Hadoop

Figure 8: Executing the command to store the parameters

Figure 9: The part-00000 file is the only file in the directory.

```
            jockey,
            course);

Console.WriteLine("{0}\t{1}", horse, stats);
    }
}
```

The reducer takes the output of the mapper, again via the console as supplied by the Hadoop framework, and reduces the list of statistics to one overall "winning index" for each horse. The reducer then emits a key value pair where the key is the horse and the value is the overall index, as in the next bit of code:

```
static void Main(string[] args)
{
string line = string.Empty;
while ((line = Console.ReadLine()) != null)
{
  string[] fields = line.Split('\t');
  string key = fields[0];
  string value = fields[1];
  string[] stats = value.Split(',');
  double index =
  stats.Sum(x => Convert.ToDouble(x));
  Console.WriteLine("{0}\t{1:.00}",
key, index);
    }
}
```

Now that it's compiled, there are a number of things to do before you can run this job on Hadoop. The first is that you have to store the input file onto the Hadoop cluster. To do this, open the Hadoop console window from the link installed to your desktop and use the Hadoop **fs –put** command, as seen in **Figure 5**.

Now you need to push the Map and Reduce functions up to Hadoop too, as seen in **Figure 6** and **Figure 7**.

Now all you have to do is start this job using the Hadoop **jar** command. You use this command because the streaming API is defined as a **jar file**. The command has a number of parameters that tell Hadoop:

- **Files**: Where the mapper and reducer files are stored
- **Mapper**: Which file is the mapper
- **Reducer**: Which file is the reducer
- **Input**: Where the input file is
- **Output**: Where to put the output from the reducer

The command is executed as in **Figure 8**.

2When the command has finished, there will be a **part-n** file in the output directory. In this example, it's called **/demo/out.** There's one file for each reducer. Because you're running the emulator, there will only be one reducer and so there will only be one file in the directory, **part-00000**, as shown in **Figure 9**.

If you cat that file, you can see the output of the algorithm, which is each horse's name and its calculated "winning index," as in **Figure 10**.

Listing 2: Define a mapper

```
public class Mapper : MapperBase
{
    public override void Map(string
      inputLine,
      MapperContext context)
    {
        string[] fields = inputLine.Split(',');
        string horse = fields[0];
        string jockey = fields[1];
        string course = fields[2];

        string stats = GetStatsForHorseJockeyCourse(
            horse,
            jockey,
            course);

        context.EmitKeyValue(horse, stats);
    }

    private static string GetStatsForHorseJockeyCourse(
        string horse,
        string jockey,
        string course)
    {
        // Your code here J
}
```

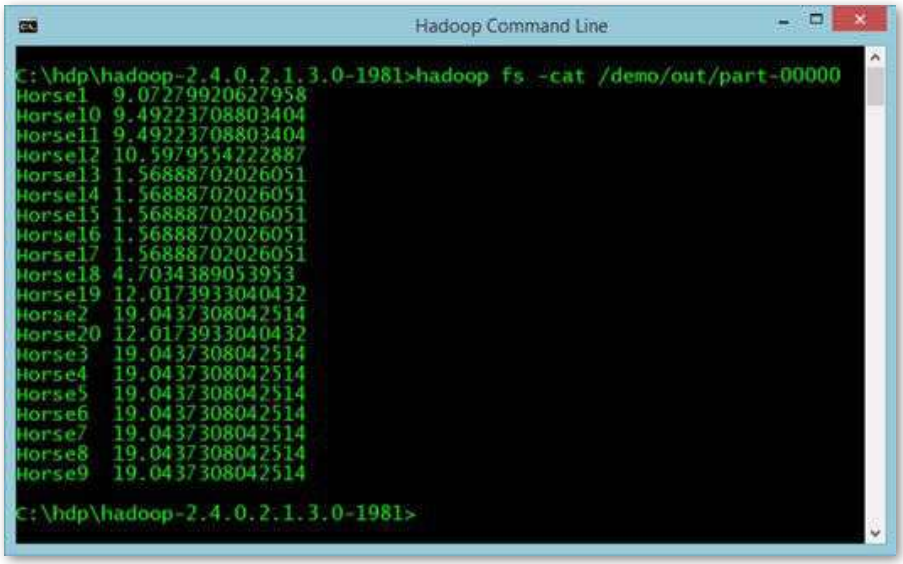


Figure 10: Each horse's winning index is calculated.

The Hadoop.MapReduce Package

I won't lie, that last exercise is a little long-winded. Fortunately, there's an easier way to work with Hadoop and that's via the Microsoft.Hadoop.MapReduce package. Once that package is added to your project, it becomes much easier to work with Hadoop.

There are only three things you have to do once you've added the package. The first of these is to define a mapper, as shown in **Listing 2.**

The only difference between the code in **Listing 2** and the code defined for the streaming API version is that this class inherits from MapperBase.

The next thing you have to do, not surprisingly, is to define a reducer:

```
public class Reducer : ReducerCombinerBase
{
    public override void Reduce(
        string key,
        IEnumerable<string> values,
        ReducerCombinerContext context)
    {
        string value = values
            .First()
            .Split(',')
            .Sum(x => Convert.ToDouble(x))
            .ToString();

        context.EmitKeyValue(key, value);
    }
}
```

Again, the only difference between this version of the reducer and the previous version is that this one inherits from **ReducerCombinerBase**. The purpose of a combiner is outside the scope of this article; I'm not defining a combiner in the example.

The last thing you have to do is define the Hadoop job, which can be seen in the following snippet:

```
class Program
{
    static void Main(string[] args)
    {
    HadoopJobConfiguration conf
      = new HadoopJobConfiguration()
        {
            InputPath = "/demo/in",
            OutputFolder = "/demo/out"
        };

       Hadoop.Connect(new Uri("http://YOUR_MACHINE_NAME/"),
  YOUR_UID, YOUR_PWD)
            .MapReduceJob
            .Execute<Mapper, Reducer>(conf);

    }
}
```

Here, all you're doing is instantiating a HadoopJobConfiguration object and setting the input and output directories, much like you specified in the earlier call. Now you can connect to the Hadoop emulator and execute a Map / Reduce job using the defined mapper and reducer.

I hope I've shown you that Hadoop is not just a tool in the Unix-like world and that it's usable on the Microsoft platform, which allows you to exercise the .NET skills you already have.

Gary Short

(Continued from 74)

practice, just as anything else does. Start by volunteering at your local user group.

Lesson #3: Be Reliable

"In international politics, states cooperate more readily when they believe that their partners will live up to their promises and abide by prior agreements. There's a vast literature on this topic, and one of the things it teaches is that if states want to work effectively with others, a reputation for reliability helps." This isn't just about showing up to meetings on time; this is also about delivering what we promise, in the timeframe we promise it, within the budget we promised. Reliability is a reputation earned, not just handed out.

This doesn't mean that you agree to everything put in front of you—part of that reliability is a willingness to call BS when you see it and refusing to agree to expectations that are outside of your abilities. (Notice that I refrain from using the word "impossible"—nothing is impossible, it just may not be possible for you, or me, or anyone else we know personally.) When a customer demands something you can't do, then it's up to you to say so. But it's also up to you to figure out what you can do that would meet them some part of the way.

Lesson #4: Think Strategically and Plan Ahead

"World history is filled with suffering that occurred because national leaders didn't think through what they were doing and didn't have a coherent reality-based plan of action. Germany started two world wars for foolish reasons, lost them both, and millions of people died in the process. The Soviet Union threatened world revolution and built a vast military machine, which led the world's most powerful states to join forces to contain it until it collapsed. George Bush consciously decided to invade Iraq without the slightest idea of what he would do once Saddam was defeated. These governments (and many others) allowed myths, delusions, and wishful thinking to guide their decisions, and the world paid an enormous price for their blunders."

Ah, if there is one lesson that history teaches those who would lead nations, it's that planning is important. Not just the immediate planning of "how will I accomplish this task in front of me," but planning for what will happen after that…and after that, and after that, and after that. Military staff colleges "wargame" various scenarios all the time, so that they can present the President with suggestions and predicted outcomes. In fact, this was part of the thought process that went into the Architectural Katas (https://archkatas.herokuapp.com/) when I put them together. As a developer, you need to be planning on two levels, both what you are working on and how your career will progress beyond the current day job. What do you want to do? What's your career goal? When do you want to retire? If you can't answer these basic questions, how can you know if the next job offer you receive is advancing your career or just another code-slinging gig?

Lesson #5: Choose Your Allies Carefully

And if there's any *other* lesson history can teach, it's that choosing allies can make or break a nation's survival. The United States has proven to be a good ally in two world wars, but arguably less so in Vietnam, Afghanistan, or Iraq. Italy historically has had less great a reputation. Poland discovered the USSR to be a poor treaty partner at the start of World War II. The list goes on, and on, and on.

In technology as in politics, choosing your allies—your coworkers—can make a huge difference. A good manager can make everything effortless for you. A poor manager can push your productivity to zero. And while some of us get to pick and choose coworkers, most can't. This doesn't mean neglecting them is a viable option. Your interaction with your coworkers will define how well they rally to support you when you're under the gun—and that will be more often than you might think. Their willingness to stand by you will often parrot your willingness to stand by them under similar circumstances, and the same will be true of your management. In any company larger than one person, there will always be politics—but these need not always be "negative" politics if you approach the discussions as "win-win" negotiations.

Just as most nations aren't hell-bent on psychopathic behavior, neither are most coworkers. Assume, despite the temptation to believe the contrary, that your coworkers are rational actors. They're behaving this way for a reason. What do they want? What are their goals? What resources are they craving? Find ways in which their goals and interests align with your own and push those positions. Even the crankiest coworker can become a staunch ally if you show them that you're genuinely interested in helping them achieve what they want and/or need.

Summary

International Relations isn't going to make you a better programmer, just as Computer Science isn't going to teach you how to be a better communicator. But contrary to popular opinion, both sets of skills are necessary to be an effective developer and will be for the foreseeable future, particularly if you look for opportunities to grow vertically in the industry (whether as management or as a "chief scientist" somewhere).

It's really a simple list. Embrace change. Learn to communicate well. Be reliable. Think strategically. Choose your allies well.

Maybe that liberal arts degree wasn't such a bad idea after all.

Ted Neward

CODE

Sep/Oct 2015
Volume 16 Issue 5

Group Publisher
Markus Egger

Associate Publisher
Rick Strahl

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Content Editor
Melanie Spiller

Writers In This Issue
Jason Bender, Kevin S. Goff, Ted Neward, Walt Ritscher, Gary Short, Markus Egger, Sahil Malik, John Petersen, Paul Sheriff, Mike Yeager

Technical Reviewers
Markus Egger
Rod Paddock

Art & Layout
King Laurin GmbH
info@raffeiner.bz.it

Production
Franz Wimmer
King Laurin GmbH
39057 St. Michael/Eppan, Italy

Printing
Fry Communications, Inc.
800 West Church Rd.
Mechanicsburg, PA 17055

Advertising Sales
Tammy Ferguson
832-717-4445 ext 026
tammy@codemag.com

Circulation & Distribution
General Circulation: EPS Software Corp.
International Bonded Couriers (IBC)
Newsstand: Ingram Periodicals, Inc.
Media Solutions

Subscriptions
Subscription Manager
Colleen Cade
832-717-4445 ext 028
ccade@codemag.com

US subscriptions are US $29.99 for one year. Subscriptions outside the US are US $44.99. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards accepted. Bill me option is available only for US subscriptions. Back issues are available. For subscription information, email subscriptions@codemag.com or contact customer service at 832-717-4445 ext 028.

Subscribe online at
www.codemag.com

CODE Developer Magazine
6605 Cypresswood Drive, Ste 300, Spring, Texas 77379
Phone: 832-717-4445
Fax: 832-717-4460

On Liberal Arts

For an industry that prides itself on its analytical ability and abstract mental processing, we often don't do a great job applying that mental skill to the most important element of the programmer's tool chest: ourselves. "You're a what major, again?" For many years, while interviewing for programming positions,

I had to answer this question over and over again—because if it wasn't already obvious, I wasn't a Computer Science major in college. Or any other science, in fact. My degree is a Bachelor of Arts in International Relations, graduated 1995 from the University of California at Davis (Go Aggies!). If you are surprised, dear reader, believe you me, the thought of a Liberal Arts student interviewing for programming positions in the late 90s was, somehow, only slightly more incredible to some of these HR folks than a bear wearing a tutu playing AC/DC's "Thunderstruck" on the bagpipes while on a unicycle.

Despite that entrenched resistance to non-CompSci candidates, I still managed to find a few positions willing to "take a chance" on a candidate with some apparent C++ skills, and lo and behold, before long, I had a career. And people stopped asking me about my degree in college.

All of that was more or less behind me until today, when while reading my online copy of *Foreign Policy* magazine, I happened across an editorial/opinion piece entitled "The Five-Minute Commencement Speech (or, what IR theory can teach you about living a happy and productive life)", by Stephen M Walt. (The full link is here: https://foreignpolicy.com/2015/05/20/the-five-minute-commencement-speech-international-relations-graduation) Reading through it, I felt a kinship to the author, not to mention a surge of vindication.

So, with that, indulge me a column, and allow me to explain what international relations theory can teach you, the programmer, about living a happy and productive life as a programmer.

Lesson #1: Change is the New Constant

Walt writes, "Until a few centuries ago, human society changed very slowly. It took more than 50,000 years for the world's human population to reach 1 billion, but that number has doubled three times since 1800. For most of human history, people were born, lived, and eventually died without experiencing significant social or technological change. And until a few hundred years ago, most people were unaware of the vast majority of societies that shared the planet with them. By contrast, today we live in a globalized world of mutual awareness and where the pace of change is positively dizzying. None of us know what will be humanly possible a few decades from now, or how scientific advances and diverse cultural forces will reshape attitudes, social interactions and political institutions.

"My advice: *get used to it*. *Embrace it*. IR theory tells us that globalization may wax and wane somewhat, but the pace of change is not going to slow. Nothing is forever—not vinyl records, not CDs, not the latest smartphone app—and some of our cherished notions about politics and society are headed for the dustbin of history too. Norms and beliefs and theories that millions once held sacred seem like barbaric relics to us today, and some ideas that you think are unquestionably true right now will look foolish to you a few years from now. This process is called *learning*, and if this university taught you nothing else, I hope it taught you that changing your mind is not something to fear."

I really wish I could tattoo this entire pair of paragraphs on every developer's forehead. Far more so than the world itself, our industry continues to move at breakneck pace—in fact, we're part of (if not the core of) what's driving all that change in the world. Consider this: Without Twitter or any of the other social media sites, could the Arab Spring happen?

More importantly, though, look at the pace of tech in our space. Within a decade, mobile platforms went from a "maybe, someday, we'll have something close to as cool as what they had on Star Trek" to practical reality. (Matter of fact, the tablet or phone on which you're reading Walt's article is capable of things the Star Trek writers never imagined possible.)

You think this is going to change? Embrace it. Think your chosen language will be the last one you ever program in? Think your chosen platform will exist forever? Talk to the DOS developers, the Windows 3.X developers, the COM developers, and, yes, of course, the developers who spent years working on hard-drive compression utilities.

Lesson #2: Learn to Write and Talk Well

If the current discussion around working from home and distributed companies has taught us anything about programming, it's that effective communication is fast becoming more important. Think about it: if all we really had to do as programmers is put our heads down and type furiously on the keyboard, why do so many companies seem to be making so much money off of communication tools? Trello, HipChat, Slack—the list goes on for days. Programming is clearly a communicative process, as much as some old-schoolers may want to pretend otherwise, but being able to express yourself clearly—whether in a meeting, code comments, or email—is absolutely critical.

Brace yourself—bad news is coming, and you probably already know what it is: Most of us are terrible at it. Matter of fact, let's be honest, many programmers got into programming in the first place to avoid having to talk to people. More than once I've had a developer tell me they prefer computers because computers are deterministic creatures and people are not. Communicating with other people is hard work.

Here's the good news: this is a fixable problem. It doesn't take a ton of work, but it will take time and practice. Just like learning a new programming language, it begins with reading, and it happens through practice. Read about writing; Strunk and White is the Kernighan and Ritchie of writing, the place everybody begins. Pretty much anything by Stephen Pinker is a good follow up. As a matter of opinion, just about any non-technical non-fiction book is a good follow-up. Read. Examine why the writer seems able to communicate the clarity of his thoughts—how did she approach the subject? What was the organization?

Start a blog. Don't settle for poor grammar or typographical errors. Ask a friend to copyedit, or find an editor online somewhere and pay them (!) to teach you what you're getting wrong.

The same goes for public speaking. Sure, programming is a communion of programmer and compiler/interpreter, but then you have to stand in front of a room of other developers and customers and explain what you did. Or why what you did works better than what they wanted you to do. Or why their million-dollar idea is entirely irrational. Wouldn't you prefer to be someone who can command the room's respect with your speaking skills while trying to tell them that they're all idiots? You don't need to be Scott Guthrie or Scott Hanselman—because (sssshhhh) they didn't used to be those guys, either. It takes

(Continued on page 73)

SXSW INTERACTIVE 2016

MARCH 11–15
AUSTIN, TX

REGISTER TO ATTEND

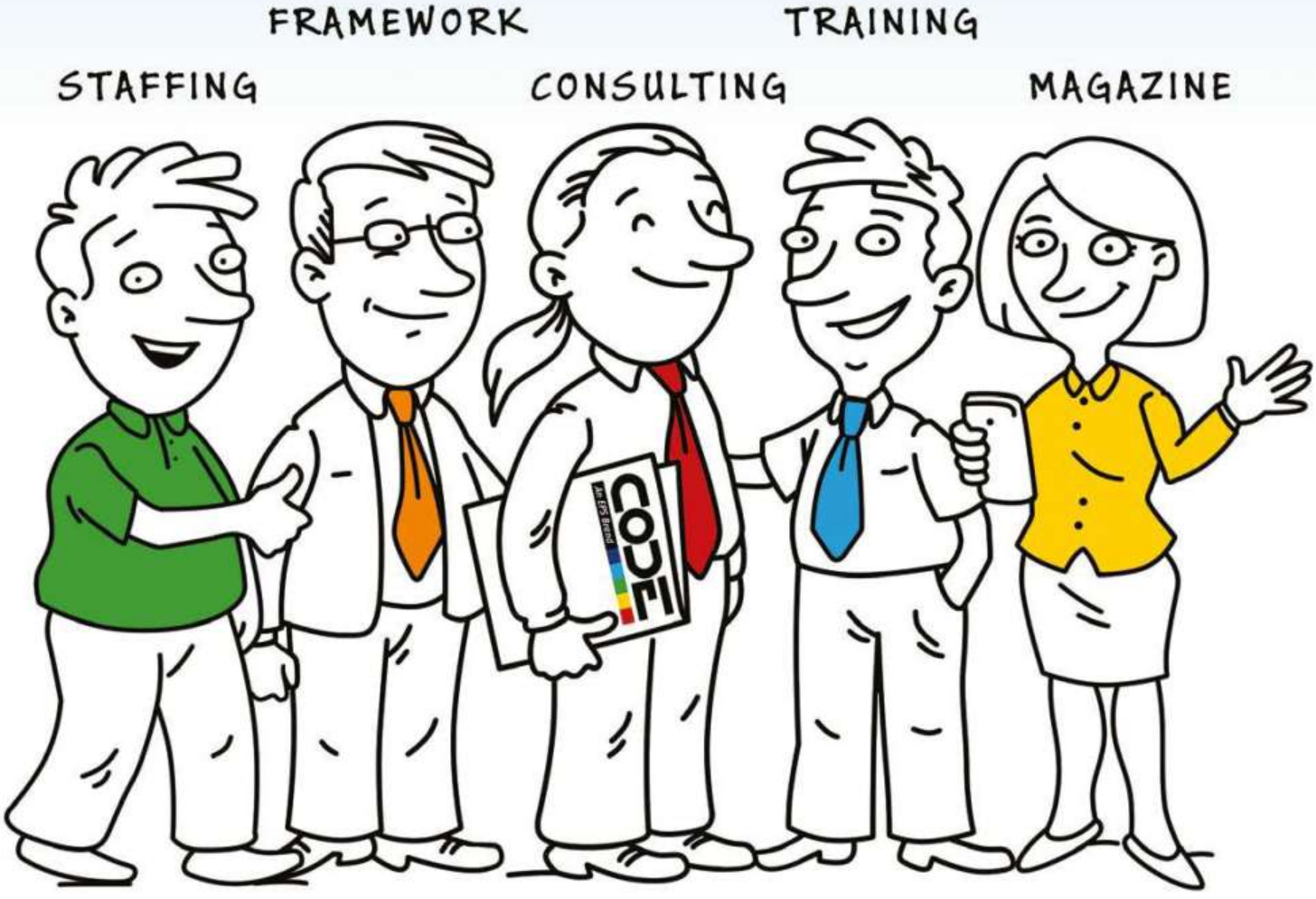
SXSW.COM/ATTEND

PANELPICKER NOW OPEN!

SXSW 2016 PanelPicker voting ends on Friday, September 4.

SXSW.COM/INTERACTIVE

CODE - More Than Just CODE Magazine



The CODE brand is widely-recognized for our ability to use modern technologies to help companies build better software. CODE is comprised of five divisions - CODE Consulting, CODE Staffing, CODE Framework, CODE Training, and CODE Magazine. With expert developers, a repeatable process, and a solid infrastructure, we will exceed your expectations. But don't just take our word for it - ask around the community and check our references. We know you'll be impressed.

Contact us for your free 1-hour consultation.

Helping Companies Build Better Software Since 1993

www.codemag.com
832-717-4445 ext. 9 • info@codemag.com

CODE
An EPS Brand